Calhoun: The NPS Institutional Archive

DSpace Repository

Theses and Dissertations | 1. Thesis and Dissertation Collection, all items

1991-09

Monitoring of global acoustic transmissions: signal processing and preliminary data analysis.

Frogner, Gary Russell

Monterey, California. Naval Postgraduate School

http://hdl.handle.net/10945/28379

Downloaded from NPS Archive: Calhoun

http://www.nps.edu/library

Calhoun is the Naval Postgraduate School's public access digital repository for research materials and institutional publications created by the NPS community. Calhoun is named for Professor of Mathematics Guy K. Calhoun, NPS's first appointed -- and published -- scholarly author.

Dudley Knox Library / Naval Postgraduate School
411 Dyer Road / 1 University Circle
Monterey, California USA 93943

LIBRARY
NAVAL POSTGRADUATE SCHOOL
MONTEREY, CALIFORNIA 93943-5002

NAVAL POSTGRADUATE SCHOOL

Monterey, California

THESIS

MONITORING OF GLOBAL ACOUSTIC TRANSMISSIONS: SIGNAL PROCESSING AND PRELIMINARY DATA ANALYSIS

by

Gary Russell Frogner

September, 1991

Thesis Advisor: James H. Miller
Co-Advisor: Ching-Sang Chiu

Approved for public release; distribution is unlimited

T259756

Unclassified

Security Classification of this page

REPORT DOCUMENTATION PAGE

1a Report Security Classification: Unclassified		1b Restrictive Markings	
2a Security Classification Authority		3 Distribution Availability of Report: Approved for public release; distribution is unlimited	
2b Declassification/Downgrading Schedule			
4 Performing Organization Report Number(s)		5 Monitoring Organization Report Number(s)	
6a Name of Performing Organization: Naval Postgraduate School	6b Office Symbol (*If Applicable*): 35	7a Name of Monitoring Organization: Naval Postgraduate School	
6c Address (*city, state, and ZIP code*): Monterey, CA 93943-5000		7b Address (*city, state, and ZIP code*): Monterey, CA 93943-5000	
8a Name of Funding/Sponsoring Organization	8b Office Symbol (*If Applicable*)	9 Procurement Instrument Identification Number	
8c Address (*city, state, and ZIP code*)		10 Source of Funding Numbers	
		Program Element Number	Project No / Task No / Work Unit Accession No

11 Title (*Include Security Classification*): MONITORING OF GLOBAL ACOUSTIC TRANSMISSIONS: SIGNAL PROCESSING AND PRELIMINARY DATA ANALYSIS

12 Personal Author(s): Frogner, Gary R.

13a Type of Report	13b Time Covered	14 Date of Report (*year, month,day*)	15 Page Count
Master's Thesis	From To	September 1991	137

16 Supplementary Notation: The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

17 Cosati Codes			18 Subject Terms (*continue on reverse if necessary and identify by block number*)
Field	Group	Subgroup	Acoustic Tomography, Hadamard Transform, Heard Island Experiment

19 Abstract (*continue on reverse if necessary and identify by block number*)

A great deal of controversy exists concerning the possible global warming trend which may occur as a result of a documented increase in atmospheric "greenhouse" gasses. The 1991 Heard Island Feasibility Experiment tested the feasibility of using transmissions of acoustic energy through the major ocean basins of the world to monitor spatially averaged global temperature trends. This thesis documents the Naval Postgraduate Schools reception of the phase encoded signal transmitted from the Southern Indian Ocean, development of real-tme signal processing software, and preliminary data analysis.

Data, received from a 32-channel vertical array suspended in the deep sound channel off the coast of Monterey, CA, was processed using real-time capable software. Data processing to reduce noise, determine SNR, and remove the m-sequence coding was found to be quite sensitive to Doppler frequency shifts. Although the SNR of the raw data was only about -27.5 dB for individual hydrophones, the transmitted signal was detected in both the frequency and time domains. However, the maximum processed signal peak in the time domain had an SNR of only +9 dB which is insufficient for use in a long term global temperature monitoring project, as it provides inadequate arrival time resolution.

20 Distribution/Availability of Abstract	21 Abstract Security Classification	
[X] unclassified/unlimited [ ] same as report [ ] DTIC users	Unclassified	
22a Name of Responsible Individual: James H. Miller	22b Telephone (*Include Area code*): (408) 646-2384	22c Office Symbol: EC/Mr

DD FORM 1473, JUN 86 *Previous editions are obselete.* S/N 0102-LF-014-6603

SECURITY CLASSIFICATION OF THIS PAGE: Unclassified

Approved for public release; distribution is unlimited.

Monitoring of Global Acoustic Transmissions: Signal Processing and Preliminary Data Analysis

by

Gary Russell Frogner
Lieutenant Commander, U.S. Navy
B.S.E. University of Washington, 1981

Submitted in partial fulfillment of the requirements for
the degree of

MASTER OF SCIENCE IN PHYSICAL OCEANOGRAPHY

from the

NAVAL POSTGRADUATE SCHOOL
September 1991

ABSTRACT

A great deal of controversy exists concerning the possible global warming trend which may occur as a result of a documented increase in atmospheric "greenhouse" gasses. The 1991 Heard Island Feasibility Experiment tested the feasibility of using transmissions of acoustic energy through the major ocean basins of the world to monitor spatially averaged global temperature trends. This thesis documents the Naval Postgraduate School's reception of the phase encoded signal transmitted from the Southern Indian Ocean, development of real-time signal processing software, and preliminary data analysis.

Data, received from a 32-channel vertical array suspended in the deep sound channel off the coast of Monterey, CA, was processed using real-time capable software. Data processing to reduce noise, determine SNR, and remove the m-sequence coding was found to be quite sensitive to Doppler frequency shifts. Although the SNR of the raw data was only about -27.5 dB for individual hydrophones, the transmitted signal was detected in both the frequency and time domains. However, the maximum processed signal peak in the time domain had an SNR of only +9 dB which is insufficient for use in a long term global temperature monitoring project, as it provides inadequate arrival time resolution.

TABLE OF CONTENTS

I. INTRODUCTION 1

A. GLOBAL ACOUSTIC TRANSMISSIONS 1

B. THE 1991 HEARD ISLAND FEASIBILITY EXPERIMENT 5

1. Objectives 5

2. Importance 6

C. THESIS OBJECTIVES 7

D. THESIS ORGANIZATION 8

II. BACKGROUND 9

A. THE 1991 HEARD ISLAND FEASIBILITY EXPERIMENT 9

1. Scope and Objectives 9

2. Transmission 10

3. Reception 13

B. ACOUSTIC SIGNAL PROPAGATION 16

1. Propagation Theory 16

2. Propagation Losses 22

a. Spreading Losses 22

b. Absorption Losses 24

3. Signal-to Noise Ratio 25

C. MAXIMAL LENGTH SEQUENCES 26

1. Introduction 26

2. M-Sequence Generation 28

3. Characteristics 29

4. Autocorrelation Process 33

5. The Hadamard Matrix 35

6. The Fast Hadamard Transform 38

7. Vector Order Permutation 41

8. Correlation Summary 41

III. SIGNAL PROCESSING 42

A. SIGNAL SPECIFICATION 42

1. Continuous Wave 42

2. 3/10 Pentaline 42

3. Maximal Length Sequences 44

B. DATA COLLECTION AND PRE-PROCESSING 44

1. Amplification 44

2. Analog Filtering 45

3. Analog-to-Digital Conversion 45

4. Clipping 47

5. Doppler Correction 47

C. SIGNAL PROCESSING PROCEDURE 50
1. General Considerations 50
2. Bandpass Filter Module 52
3. Demodulation 53
4. Low Pass Filter 54
5. Sequence Removal 55
D. POST-PROCESSING ENHANCEMENTS 58
1. Time Averaging 58
2. Visual Integration 58

IV. RESULTS 60
A. DATA COLLECTION 60
B. SIGNAL PROCESSING 62
1. Objective 62
2. Test Data 63
3. Resampling and Clipping 63
4. Bandpass Filtering 65
5. Demodulation 65
6. Lowpass Filtering 66
7. Sequence Removal 68
8. Averaging and Display 71
C. PRELIMINARY DATA ANALYSIS 73

1. Objective 73

2. Frequency Domain Processing 73

a. Autospectral Density 73

b. Signal-to-Noise Ratio 75

c. Arrival Structure 77

3. Time Domain Processing 78

V. CONCLUSIONS AND RECOMMENDATIONS 85

A. SUMMARY OF RESULTS 85

B. CONCLUSIONS 86

1. Signal Processing 86

2. Heard Island Feasibility Experiment 87

C. RECOMMENDATIONS FOR ADDITIONAL WORK 88

APPENDIX A 90

Appendix B 91

REFERENCES 123

INITIAL DISTRIBUTION LIST 126

ACKNOWLEDGMENTS

I would be remiss indeed if I did not recognize the tremendous amount of assistance from those who contributed to this thesis. I would like to express my gratitude to Dr. Keith Von der Heydt whose expertise enabled our successful participation in the experiment. I am also indebted to Dr. Kurt Metzger for his helpful suggestions in implementing his sequence removal routine to form the core of my processing software.

I must extend a special thanks to my advisor, Professor James H. Miller, whose patience and insight into signal processing techniques have greatly expanded my understanding of acoustic tomography. I would also like to thank my co-advisor, Professor Ching-Sang Chiu, for his guidance and illumination of systematic methods of inquiry. It has been exhilarating to participate with this cohesive research team on a project of such potential to the oceanographic community as well as the environment of the planet.

I am especially appreciative of my wife, Cindy, and daughters, Melody and Emily, who encouraged my efforts and endured my absences during all stages of this thesis. Finally, I must acknowledge the lifetime of inspiration and strength I derive from my parents Louis and Joyce Frogner.

I. INTRODUCTION

A. GLOBAL ACOUSTIC TRANSMISSIONS

The widely discussed theory of greenhouse warming suggests that increasing levels of certain atmospheric gasses, such as carbon dioxide and methane, will ultimately lead to climatological warming on a global scale. Well documented increases in atmospheric carbon dioxide levels from air (Keeling and others, 1989) and ice core samples (Neftel and others, 1985) are shown in Figure 1.1.

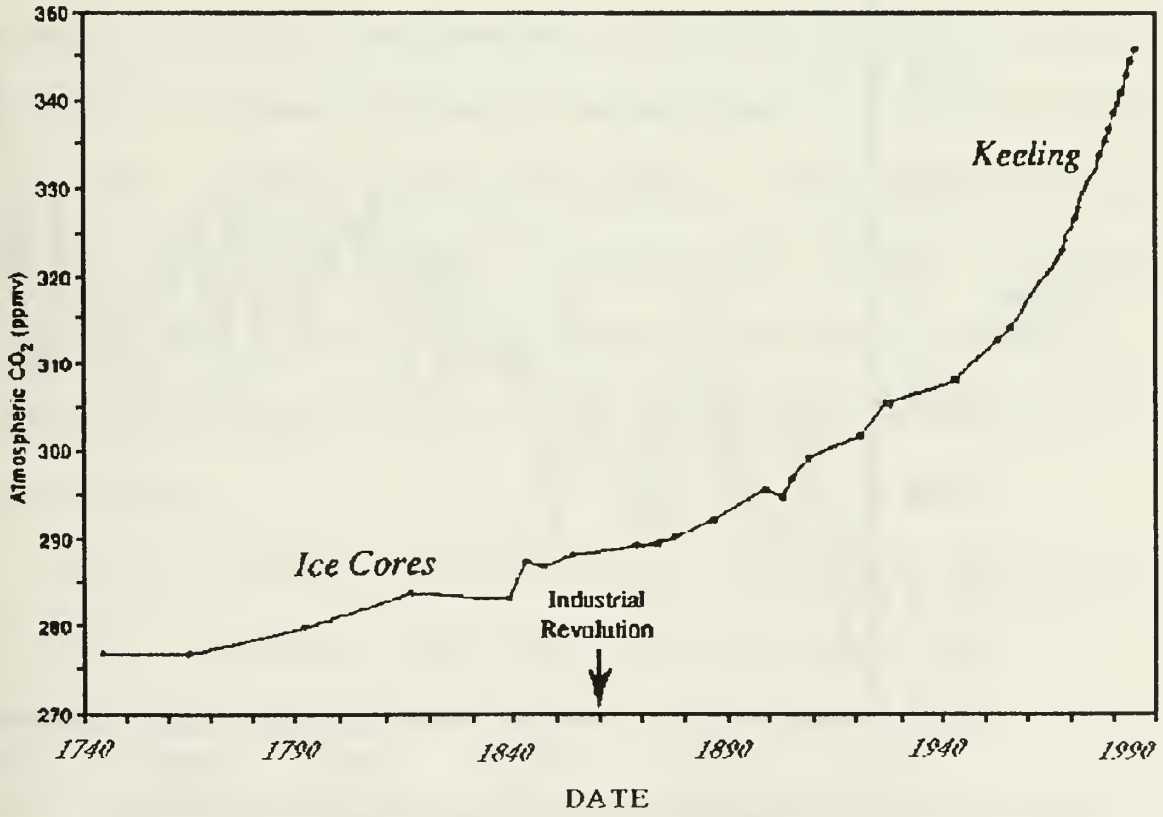

Figure 1.1: Atmospheric CO^2 Levels (Keeling 1989 and Neftel 1985).

Studies of global temperature trends, including extensive numerical modeling, indicate a possible correlation between the increase of these gasses and planetary warming. In one study (Hansen and Lebedeff, 1987) the historical atmospheric surface temperature records over the last century were compiled with the averaged results displayed in Figure 1.2. In addition, the Carbon Dioxide Assessment Committee of the National Research Council has surveyed the results of numerous global circulation models to project a continued warming trend ranging from 1 to 5 degrees Celsius within the next 50 years (NRC, 1983). This magnitude of temperature change could have dramatic consequences for the world ecosystem with significant environmental, economic, and

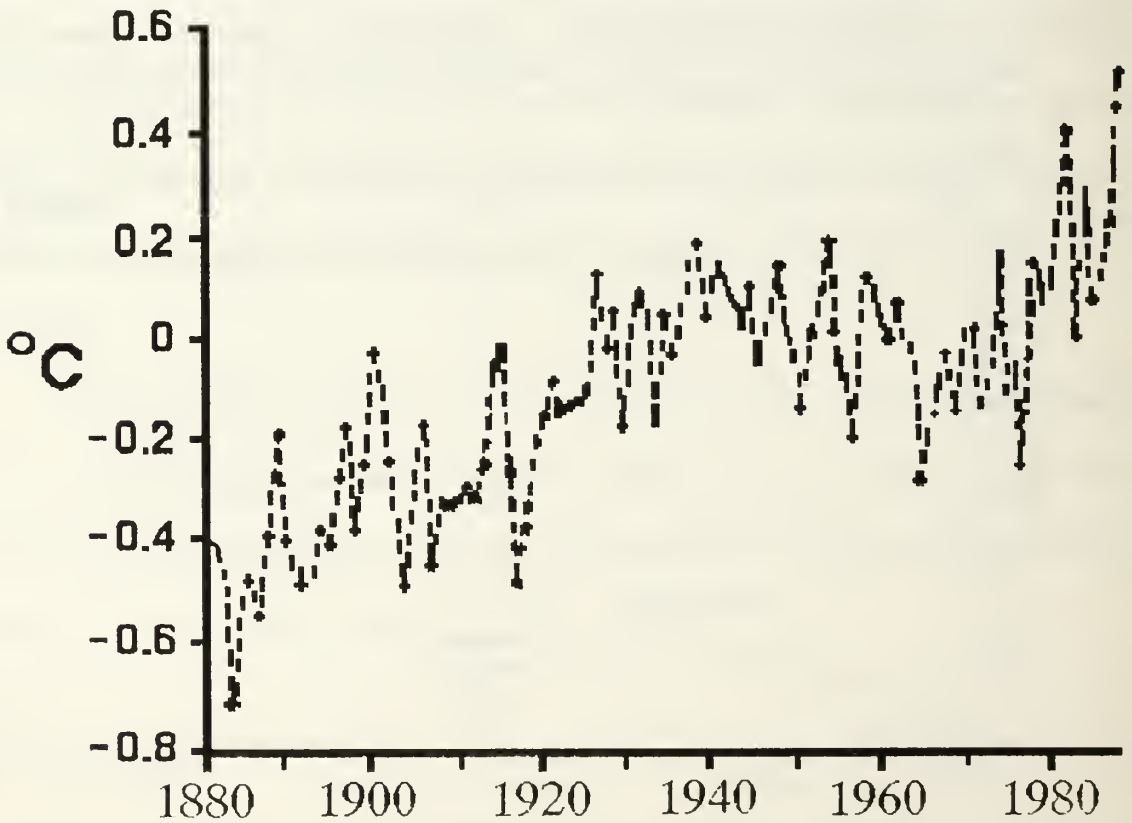

Figure 1.2: Global Temperature Trend (Hansen and Lebedeff, 1987).

political ramifications. Understandably, greenhouse warming is the subject of considerable scientific and political debate, as numerous challenging questions remain unanswered concerning the magnitude, localization, and rate of warming. However, since addressing the problem of global warming could involve huge expenditures of human resources, prudent planning and allocation of limited assets requires a more precise understanding of the coupling between the "greenhouse" gas concentration and planetary temperatures.

A reconstruction of global air temperature records over the past 108 years, shown in Figure 1.2, indicates an overall temperature increase of 0.5 °C and a 1990 warming rate of 0.02 °C per year. However, in order to statistically detect a trend in observed temperature data at a 95% confidence level, a time series long enough to produce a change in the actual mean of four times the standard deviation, σ, is required (Munk and Forbes, 1989). The Keeling CO_2 data shown in Figure 1.1 clearly shows this long term trend with over 40 σ over 33 years. Due to the inherent variability of atmospheric temperatures, the 4 σ statistical criteria would require approximately 100 years of air temperature measurements to obtain a 95 percent certainty that the earth's climate is indeed changing.

Munk and Forbes have suggested obtaining a time series of integrated ocean temperatures which, because of their smaller variance and direct coupling with the atmosphere, could produce the same degree of certainty in only 10 years. They have proposed measuring the increase in sound speed through all major ocean basins using global acoustic transmissions of low frequency sound. The highly efficient waveguide

which results from the oceanic sound speed minimum near a depth of 1 km at lower and mid-latitudes gives reasonable expectation of global sound propagation from a strong acoustic source. Since temperature is the dominant influence on the speed of sound in seawater (increasing by 5 m/s/°C), the resulting travel time change would provide a measure of the mean temperature change along the acoustic path.

Predicted oceanic warming of 0.02 °C/year at the surface and 0.004 °C at the sound channel axis would decrease the travel time over a 15,000 km range by 0.25 seconds per year (Munk and Forbes, 1989). This reduction of travel time would result in a 4 σ change (2.5 °C) in about 10 years. Figure 1.3 shows the expected decrease in travel time from the Princeton model of greenhouse warming (straight line) and a multi-level primitive-equation mode l output of mesoscale variability (Manabe and Stouffer, 1986 and Semtner and Chervin, 1990). The planetary scale of the transmissions would smooth out any localized temperature variations associated with mesoscale eddies to provide

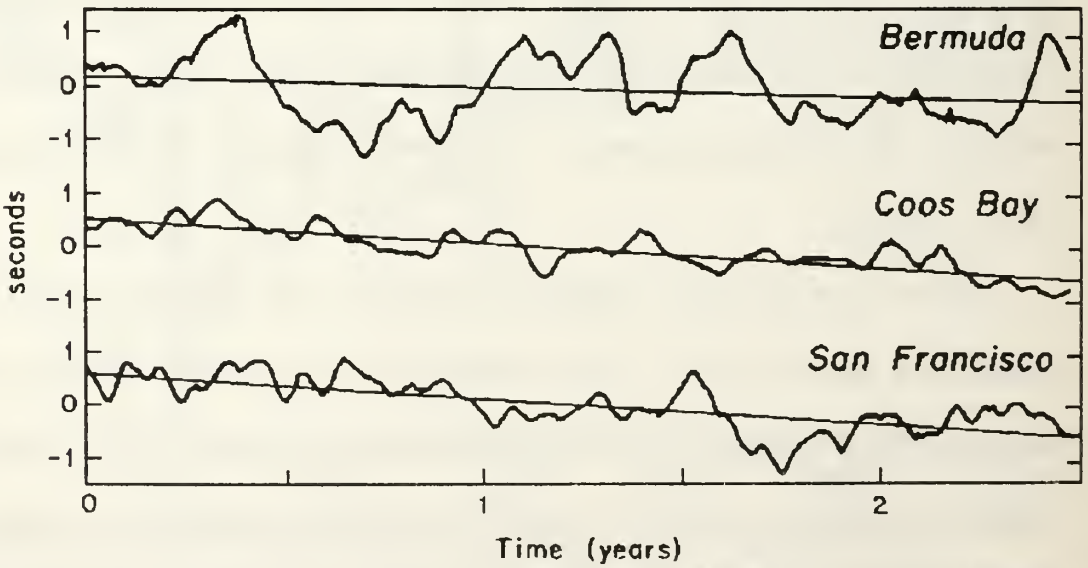

Figure 1.3: Signal Travel Time vs Calendar Years (Munk and Forbes, 1989).

indications of gyre and basin scale warming. The average travel time of the acoustic signal would thus provide an indication of the integrated temperature for each travel path.

B. THE 1991 HEARD ISLAND FEASIBILITY EXPERIMENT

1. Objectives

In January of 1991 scientists representing 9 nations and 13 universities and research institutions embarked on a worldwide experiment to test the feasibility of using acoustic transmissions to monitor the average planetary temperature. An array of powerful acoustic transducers was suspended in the sound channel approximately 40 miles east of Heard Island in the Southern Indian Ocean. The total output power of the array was approximately 213 dB referenced to 1 μPa at 1 meter. Hydrophone receivers as shown in Figure 1.4 were deployed at the following sites:

LOCATION	TYPE	INSTITUTE, COUNTRY
Ascension Island	Bottom	NOAA, USA
Tasman Sea	Towed & Sonobuoy	U.Auk., N.Zealand
Tasmania	Sonobuoys	CSIRO, Australia
Christmas Island	Sonobuoys	CSIRO, Australia
Mawson Station	Sonobuoys	CSIRO, Australia
Kerguelen Island	Sonobuoys	INSU-TAAF, France
Indian Ocean	Sonobuoys	NIO, India
Capetown	Bottom	South Africa
Fiji	Sonobuoys	Japan
North Pacific	Towed Array	IOS, Canada
Monterey, CA	Vertical Array	MBARI/NPS/WHOI/ MIT, USA
Florida	Swade Array	U. Miami, USA
Bermuda	Vertical Array	SAIC, USA
Atlantic Ocean	Navy	U. Mich., USA
Pacific Ocean	Navy	U. Mich., USA

The experiment addressed three major uncertainties underlying the use of global acoustic transmissions:

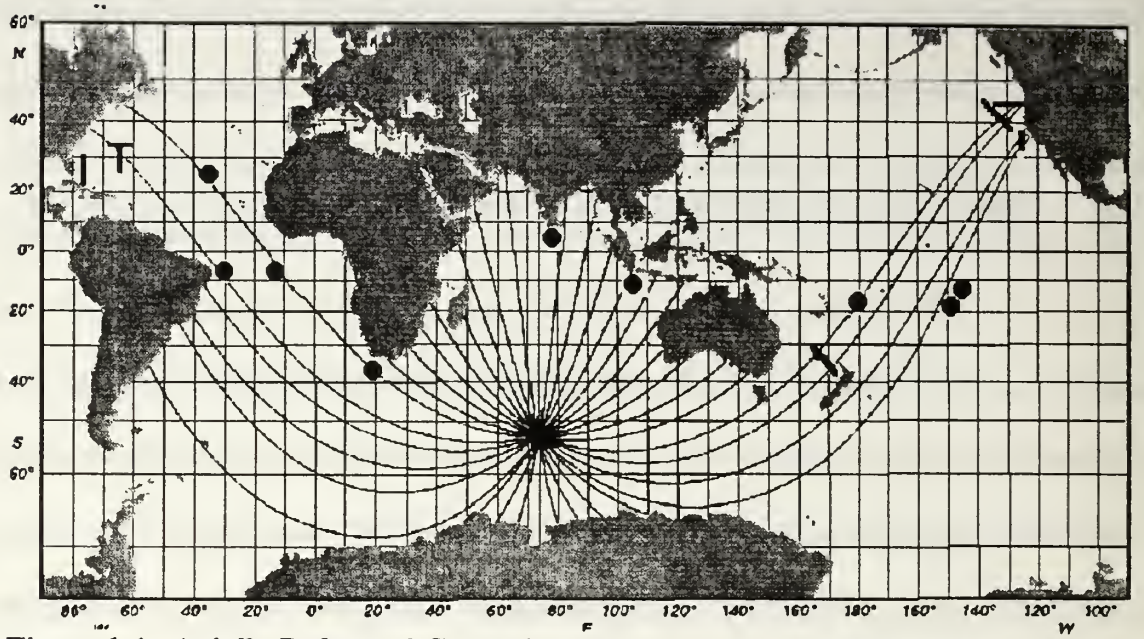

Figure 1.4: Axially Refracted Geodesics (drawn every 10° from the source) (Munk 1990).

1. The transmitted signal strength required to produce adequate signal-to-noise ratio at the receiving stations, including the achievable accuracy of arrival time measurement,

2. The stability of the multipath arrival pattern, including the maximum coherent integration times, and

3. The optimum receiver locations for a possible permanent follow-on installation.

2. Importance

Successful conclusion of the Heard Island Feasibility Experiment entails demonstration of reliable propagation of low frequency acoustic energy along global-scale transmission paths. Fruitful results could lead to the installation of a permanent climate monitoring system which could provide early warning of upper ocean warming of the Southern Ocean and interior warming in other oceans. Such a capability could provide

definitive data on the question of global warming with apparent scientific, social, and economic value.

Since the travel time of a basin-scale transmission is directly influenced by the water characteristics along its path, the arrival structure would contain information on the meso and gyre-scale processes through which the signal transited. A permanent system could also provide statistical information on the variability of these processes which could be used for calibration of operational eddy resolving global circulation models to enhance their reliability.

Tactical employment of the marine environment requires a broadened understanding of the effects of meso-scale effects on long-range, low-frequency acoustic propagation. Maintaining the tactical edge in an increasingly transparent ocean against targets with decreasing acoustic signatures demands exploitation of every available feature to optimum advantage. The spatial and temporal variability information contained in the arrival pattern would enhance the tactician's effort to optimally employ his vessel. In addition, the characteristics of a low-frequency three-dimensional wavefield which has propagated through meso-scale features will provide valuable information for effective development of future ASW systems.

C. THESIS OBJECTIVES

This thesis describes the United States Naval Postgraduate School's participation in the 1991 Heard Island Feasibility Experiment with an emphasis on signal processing and preliminary data analysis. The intent of this thesis was threefold:

1. The first goal was to deploy a vertical acoustic array and data collection hardware during the at-sea reception phase of The Heard Island Feasibility Experiment to digitize and store the received signal.

2. The second objective was to develop and implement signal processing software modules in the C programming language which will interface with Laboratory Workbench and the Concurrent 6400 real-time Unix data acquisition computer. Laboratory Workbench is a proprietary signal processing software product of the Concurrent Computer Corporation.

3. The third purpose of this thesis was to perform preliminary data analysis on the data collected during the at-sea phase including arrival time accuracy estimation, determination of signal to noise ratio, and arrival structure analysis.

D. THESIS ORGANIZATION

The organization of the remainder of this thesis is as follows:

Chapter II contains background information on the 1991 Heard Island Feasibility Experiment, ocean acoustic wave propagation, maximal-length shift-register sequences, Hadamard transforms, and acoustic Doppler processing.

Chapter III specifies the data collection techniques and signal processing utilized to analyze the received signal, including filtering, demodulation, M-sequence removal, data storage, and real time processing considerations.

Chapter IV details the results of the data collection phase, the software development effort, and the results of preliminary data analysis, including resolution, signal to noise ratio, maximum coherent integration time, multi-path arrival, and depth profile.

Chapter V summarizes the experimental results, presents the thesis conclusions and suggests some further efforts in analyzing the collected data.

II. BACKGROUND

A. THE 1991 HEARD ISLAND FEASIBILITY EXPERIMENT

1. Scope and Objectives

The objectives of the Heard Island Feasibility Experiment were:

1. to determine the required transmitted signal strength of a permanent installation which would be sufficient to overcome propagation losses and provide an adequate signal-to-noise ratio at the receiving stations, and

2. to determine whether the arrival pattern has features which are sharp and stable enough to determine the variable transmission time within the 10 millisecond accuracy required to make statistically useful measurements of global temperature trends.

The accuracy of ocean basin average temperature estimates derived from acoustic propagation is dependent on the resolution of the signal's arrival time. The arrival time resolution is in turn related to the bandwidth of the signal and the received signal-to-noise ratio. Van Trees discusses the calculation of this type of non-linear estimate with white Gaussian noise (Van Trees, 1968). The result is given as

$$\sigma_t = \frac{1}{B\sqrt{SNR}} \qquad (2.1)$$

with σ_t the arrival time uncertainty, B the bandwidth of the signal, and SNR the signal-to-noise ratio (Spindel, 1985). Re-arranging equation 2.1, the SNR required to determine the arrival time within σ_t is given by

$$SNR = \left(\frac{1}{B\sigma_t}\right)^2 \qquad (2.2)$$

Assuming a signal bandwidth of 11.4 Hz, the desired arrival time accuracy of 10 milliseconds would require a post-processed receiver SNR of at least 18.86 dB. The Monterey effort of the 1991 Heard Island Experiment attempted to maximize the receiver SNR by optimizing receiver locations, using phase-encoded pulse compression techniques, and by using a multi-element vertical receiving array.

2. Transmission

It is desirable that the permanent source for long term global transmissions be operated at less than 200 dB rel μPa@1m to allow reliable operation over a long period of time. For the 1991 Heard Island Feasibility Experiment, a vertical transmitting array of ten Hydroacoustics HLF4LL transducers with 12.5 foot spacing was deployed from the R/V Cory Chouest between 26 January and 1 February 1991 at 53° 15' S, and 73° 40' E, in an area approximately 40 miles east of Heard Island in the Southern Indian Ocean. The transmitter depth was operationally limited to 300 meters which mandated high latitude positioning where the sound axis is shallower. The position near Heard Island also allowed propagation through all major ocean basins. The array was deployed such that its center point coincided with the deep sound channel axis near 200 meters. The vessel was operated for minimal acceleration, generating minimum thrust to maintain steerage, thus introducing linear vice second order Doppler components into the signal. The operating radius of the R/V Cory Chouest was about 15 nautical miles which did not

affect the experiment since the objective was to measure the accuracy of arrival time and not the time of arrival itself.

Under ideal conditions, the total output power of a linear array of projectors is given by

$$P_{(total)} = P_o + 10 \log(N_T) \qquad (2.3)$$

where N_T is the number of active projectors and P_o is the output power of an individual projector. Nominally, five transducers, alternately spaced, were operated simultaneously and in phase to produce a broadside beam. The operating transducers were changed as required to prevent overheating. Each of the five active transducers optimally produced 206 dB rel μPa@1m to produce a combined output of 213 dB rel 1 μPa@1m.

A variety of 57.0 Hz continuous wave (CW) and phase encoded signals were transmitted on a fixed schedule consisting of one hour of continuous transmission followed by two hours of silence repeated for the entire test period. The transmission schedule is included in Appendix A. A carrier frequency of 57.0 Hz was chosen to avoid excessive attenuation at higher frequencies and increased shipping noise at lower frequencies. In addition, the resonance frequency of the transducers was close to 57.0 Hz. Figure 2.1 shows the relation between predicted ambient noise spectra and the transmitter frequency response.

The three types of transmitted signals, described in detail in Chapter III, are as follows:

- Continuous Wave at 57.0 Hz.

- 3/10 Pentaline. This is a phase encoded signal with five major spectral lines of various amplitude which facilitates easy estimates of signal-to-noise ratio based upon the observed frequency lines. However only minimal line resolution is possible since the power is split evenly between the signal and the carrier.

- M-Sequences. Maximal length shift register sequences, sometimes called pseudo-random noise, are phase encoded signals of various lengths which correspond to a specific code. Like pentaline, only half the power is in the signal. However, pulse compression techniques are used to provide excellent time resolution.

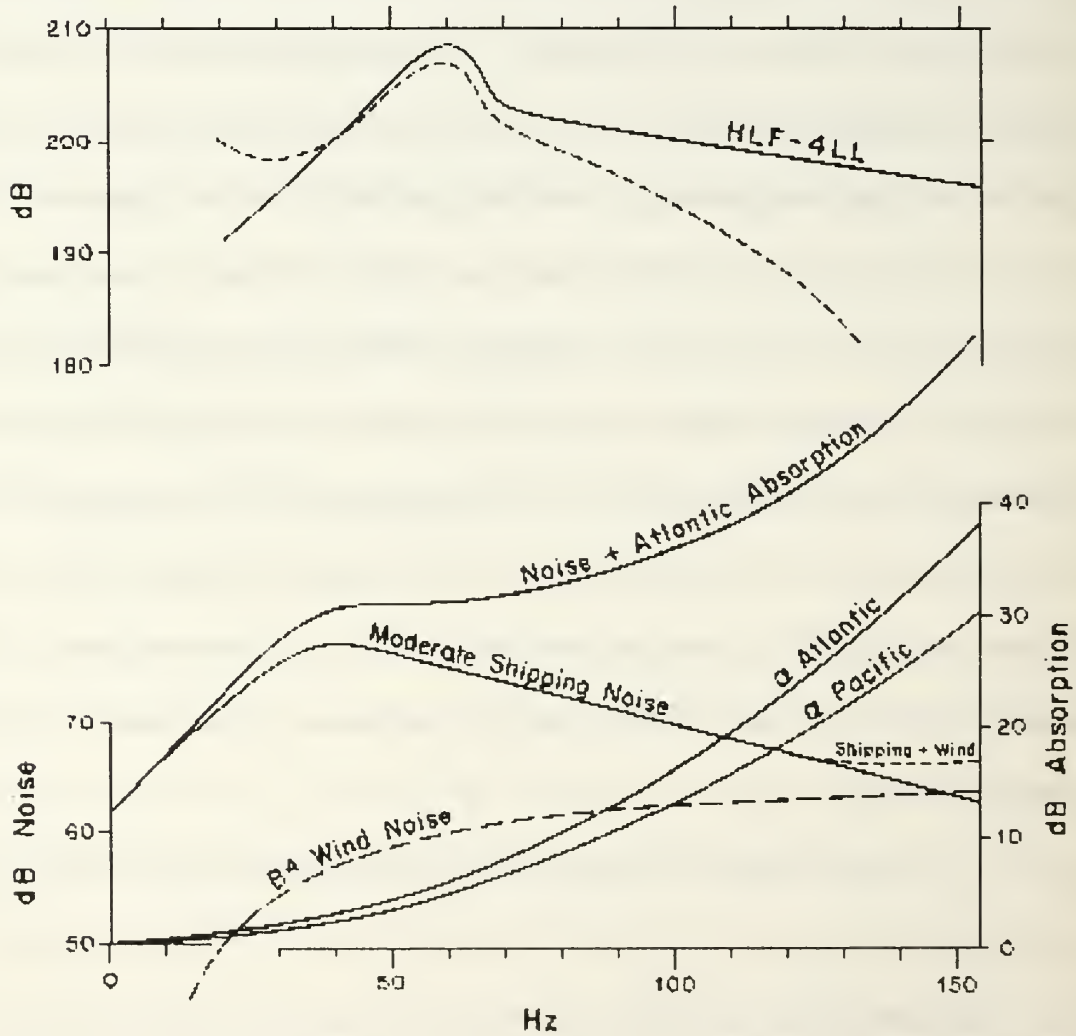

Figure 2.1: Average Ambient Noise Level and Transmitter Frequency Response [Ref. 8].

3. Reception

Since the vertical array in Bermuda suffered a disabling casualty, the array deployed jointly by the Naval Postgraduate School (NPS), Monterey Bay Aquarium Research Institute (MBARI), Massachusetts Institute of Technology (MIT), and Woods Hole Oceanographic Institute (WHOI) served as the only participating vertical array and as such will be the sole domain of this thesis. Figure 2.2 illustrates the optimal positioning of the vertical array, based on ray tracing (Ort and Chiu, 1990) utilizing a global ocean mesoscale variability model (Semtner and Chervin, 1988). This computer simulation demonstrated that acoustic rays emanating from the Heard Island transmission source at angles greater than 127° true would pass to the east of New Zealand and that rays less than 135° would escape interaction with Antarctica. In addition, these rays

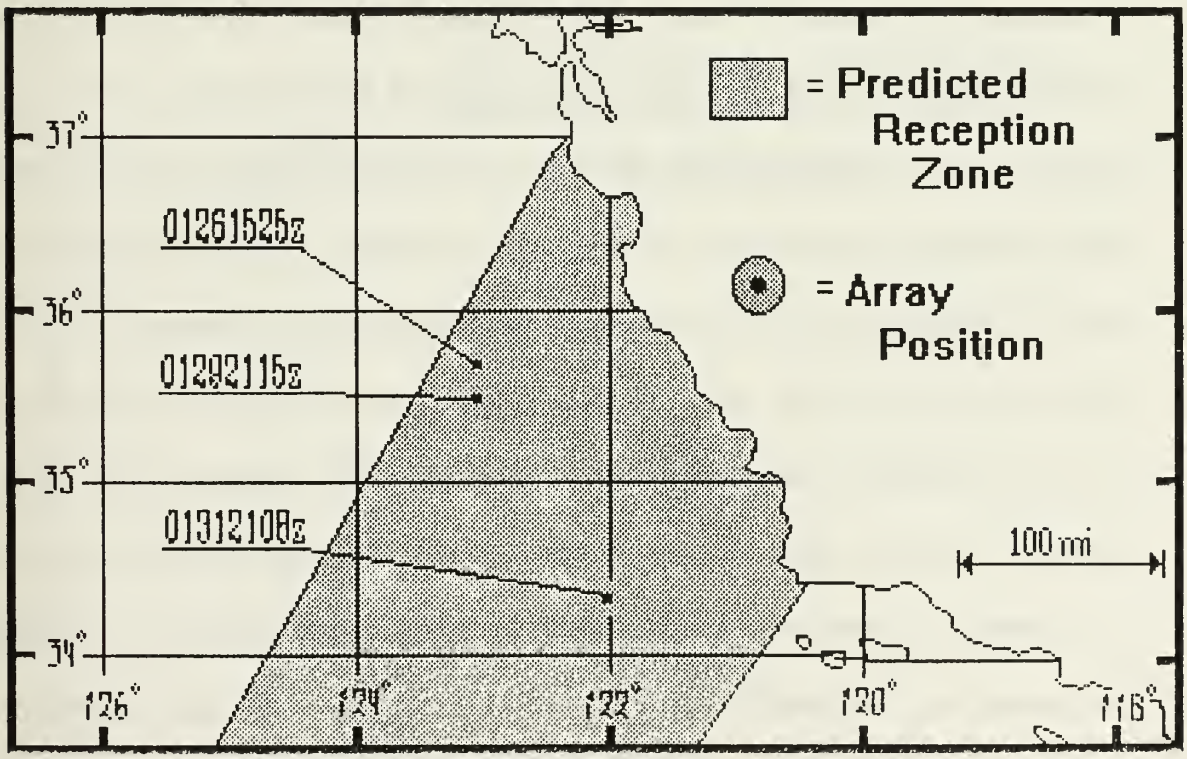

Figure 2.2: Vertical Array Position.

had five or less interactions with the ocean bottom. This band of rays, shown in Figure 2.2, intercepts the west coast of the United States in the Monterey coastal region between San Francisco and Point Conception, making it the most likely area on the U.S. west coast to receive the signal because of the convergence of rays in this vicinity.

The 32 hydrophone array was suspended vertically from *R/V Point Sur* at approximately 36° 30' N and 122° 45' E, as shown in Figure 2.3. The hydrophones were spaced 45 meters apart with the top one nominally positioned at a depth of 345 meters and the bottom one at 1740 meters. 3000 meters of kevlar core cable was used to house 50 pair of #28 solid copper conductors. The inboard 100' of cable was shielded with copper to minimize induced currents from the *R/V Point Sur*. The lower 2 km had SAIC quiet cable coating over a 28" twist reversal pattern of 0.8" per side which reduced strumming of the suspended cable. The 600 pound (submerged weight) array and its suspended ballast weight of 300 lbs. were balanced by the buoyancy from a main subsurface float and approximately half of the 30 "football" floats, spaced 2 meters apart, which isolated the array from the surface gravity wave field. The main subsurface float was a 37" sphere with 475 lbs. of buoyancy. The football floats measured 7.5" x 15" and had a buoyancy of 6.25 lbs. each. The lower weight was a 55 gallon mass attached with an acoustic release to facilitate release at recovery. The midpoint of the array was positioned with about neutral buoyancy near the center of the deep sound channel at about 1100 meters. An instrument to record PSK telemetry, pressure, array tilt, temperature, and conductivity was attached approximately 4 meters above the top hydrophone. An internally recording pressure and array tilt sensor was attached just below the lowest

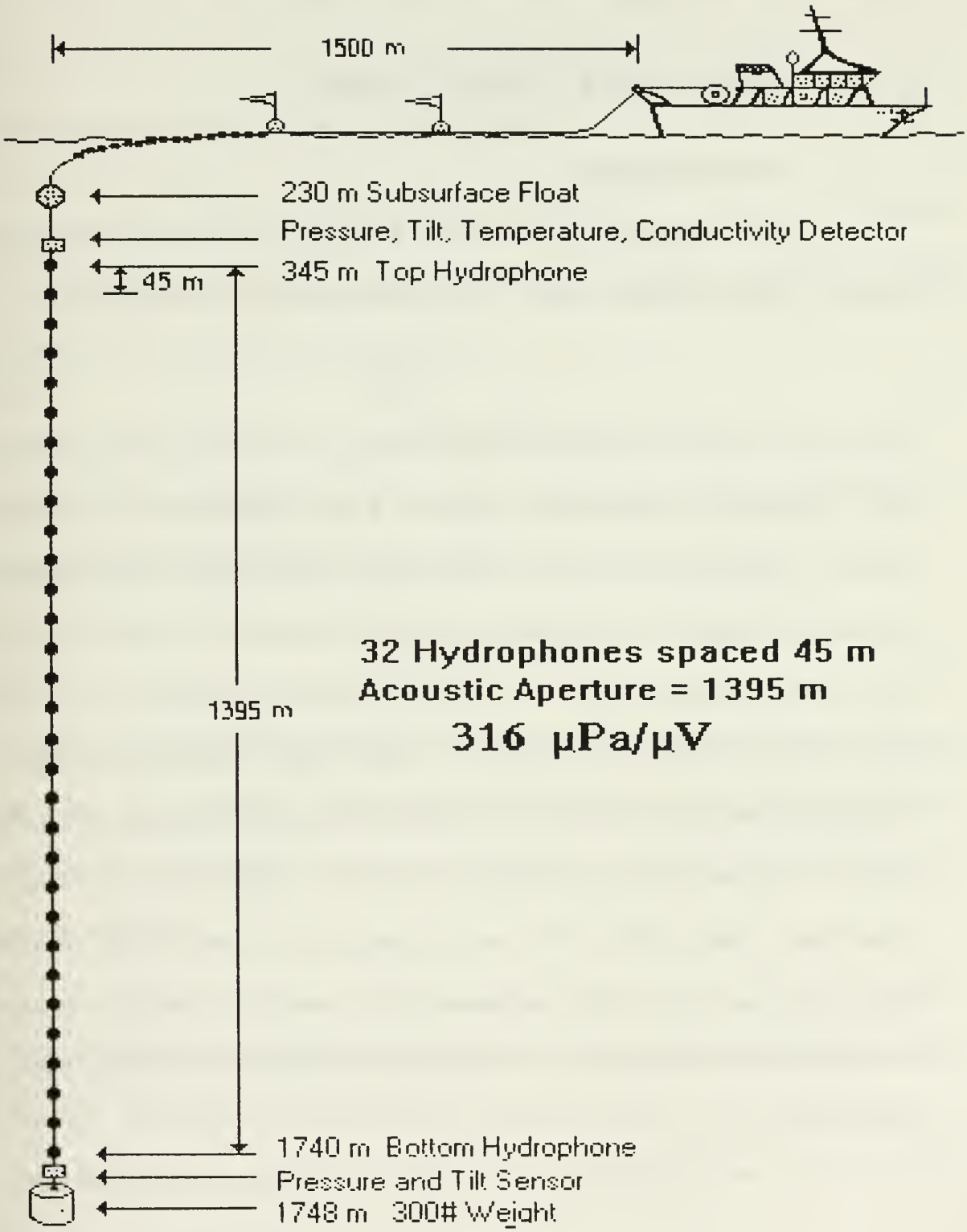

Figure 2.3: Vertical Array Configuration.

hydrophone. The submerged portion of the array was distanced from the vessel by about 1500 meters using 4 large surface floats spaced by 20 smaller floats.

B. ACOUSTIC SIGNAL PROPAGATION

1. Propagation Theory

Sound propagation in a fluid can be described by the linearized, lossless wave equation (Kinsler and others, 1982). In a motionless medium, the equation is

$$\nabla^2 p = \frac{1}{c^2} \frac{\partial^2 p}{\partial t^2} \qquad (2.4)$$

where **c(x,y,z)** is the sound phase speed field and **p(x,y,z)** is the fluid pressure disturbance field. The linearized, lossless wave equation is a good approximation in a motionless medium. Variations in the travel time due to current are generally an order of magnitude less than the changes due to deviations in the sound speed and so can be ignored in a first order treatment (Spindel, 1986). As an example, consider a mesoscale eddy in which a typical 10 cm/s surface current diminishes to about 3 cm/s below the thermocline. A corresponding temperature anomaly near the sound channel axis is approximately 1 degree Celsius. This temperature perturbation results in a sound speed perturbation of approximately 5 m/s which is 100 times the change due to current. Also note that a change in the path length (which could occur due to mesoscale variability) can have an even greater effect on the travel time, depending on the size of the change (Flatte and others, 1979).

In discussing sound wave propagation, it is useful to define a propagation vector

$$\vec{k} = k_x \vec{x} + k_y \vec{y} + k_z \vec{z} \tag{2.5}$$

that has a magnitude (or wave number) given by

$$K = \sqrt{k_x^2 + k_y^2 + k_z^2} = \frac{\omega}{c} \tag{2.6}$$

where ω is the radian frequency of the signal and **c** is the phase speed of the traveling sound wave, and a position vector given by

$$\vec{r} = x\vec{x} + y\vec{y} + z\vec{z} \tag{2.7}$$

that gives the location of the point **(x,y,z)** with respect to the origin **(0,0,0)** of a rectangular coordinate system. We can let

$$\psi = \frac{c_0 \vec{k} \cdot \vec{r}}{\omega} = \frac{\vec{k} \cdot \vec{r}}{K_o} \tag{2.8}$$

where $\mathbf{c_0}$ is a constant reference speed and Ψ has units of length. Note that if $\mathbf{c = c_0}$, then $\Psi = (\mathbf{k_x x + k_y y + k_z z}) / \mathbf{K}$ which is the distance from the origin. Surfaces of constant Ψ will now correspond to surfaces of constant phase. That is, the locus of all points **(x,y,z)** satisfying

$$\psi = constant + c_0 t \tag{2.9}$$

defines a surface of constant phase at each instant **t**, as it propagates through the acoustic medium. $\nabla\Psi$ is always perpendicular to surfaces of constant phase and is therefore the local direction of propagation of the phase surface. Substitution into the wave equation

with A(x,y,z) representing the pressure amplitude field yields

$$\left[\frac{\nabla^2 A}{A} - \left(\frac{\omega}{c_0}\right)^2 \nabla\psi\cdot\nabla\psi + \left(\frac{\omega}{c}\right)^2\right] - j\left[\frac{\omega}{c_0}\left(2\frac{\nabla A}{A}\cdot\nabla\psi + \nabla^2\psi\right)\right] = 0 \qquad \textbf{(2.10)}$$

When the real part (the value within the first square brackets) is set equal to zero, the result is the Eikonal equation. When the imaginary part (the value within the second square brackets) is set equal to zero, the Transport equation results. If **A** and $\nabla\Psi$ satisfy the inequalities

$$\left|\frac{\nabla^2 A}{A}\right| \ll \left(\frac{\omega}{c}\right)^2, \qquad |\nabla^2\psi| \ll \frac{\omega}{c}, \qquad and \quad \left|\frac{\nabla^2 A}{A}\cdot\nabla\psi\right| \ll \frac{\omega}{c} \qquad \textbf{(2.11)}$$

then all the terms in Eq. 2.10 except the second and third can be neglected and the Eikonal equation can be approximated by

$$\nabla\psi\cdot\nabla\psi = \left(\frac{c_0}{c}\right)^2 = n^2 \qquad \textbf{(2.12)}$$

where **n(x,y,z)** is the refractive index as a function of position. This is known as the plane wave approximation of the wave equation. The inequalities of Eq. 2.11 can be described as (Kinsler and others, 1982)

1. The amplitude of the wave must not change appreciably in distances comparable to the wavelength.

2. The speed of sound must not change appreciably in distances comparable to the wavelength.

3. The channel thickness and source-receiver distance must be large in comparison to a wavelength.

The 57.0 Hz signal used in this experiment had a wavelength of about 26.3 meters which satisfies all the above conditions and can thus be described using the plane wave approximation. A solution to the plane wave equation can be expressed as

$$p = A\, e^{j(\omega t - \vec{k}\cdot\vec{r})} \tag{2.13}$$

The essence of ray theory is that $\nabla\Psi$ defines the direction of wave propagation as a function of position. Therefore a point by point solution of $\nabla\Psi$ renders the path traversed by the acoustic energy. This technique has been implemented with the Hamiltonian raytracing program HARPO (Jones and others, 1986) and interfaced with output from a global eddy resolving ocean general circulation model (Semtner and Chervin, 1990), to predict the optimum receiver locations for the Heard Island Feasibility Experiment (Chiu and Ort, 1990). Typical three dimensional results are illustrated horizontally in Figure 1.4 and vertically in Figure 2.4. The rays emanating from the Heard Island source at true horizontal angles between 127° and 135° were seen to avoid significant interaction with the bottom, land masses, and seamounts that extend into the sound channel. These propagating rays can be divided into two general classifications depending on their behavior in the vertical plane.

Type 1 rays, shown in Figure 2.4(a), are trapped close to the surface by a shallow surface duct until the duct weakens at ranges between 7500 and 8000 km. The rays then enter the deep sound channel. Because the distance between the surface duct and the deep sound channel is almost 1200 m, the rays enter the deep sound channel at a relatively steep angle and then oscillate about the axis with a large spatial amplitude.

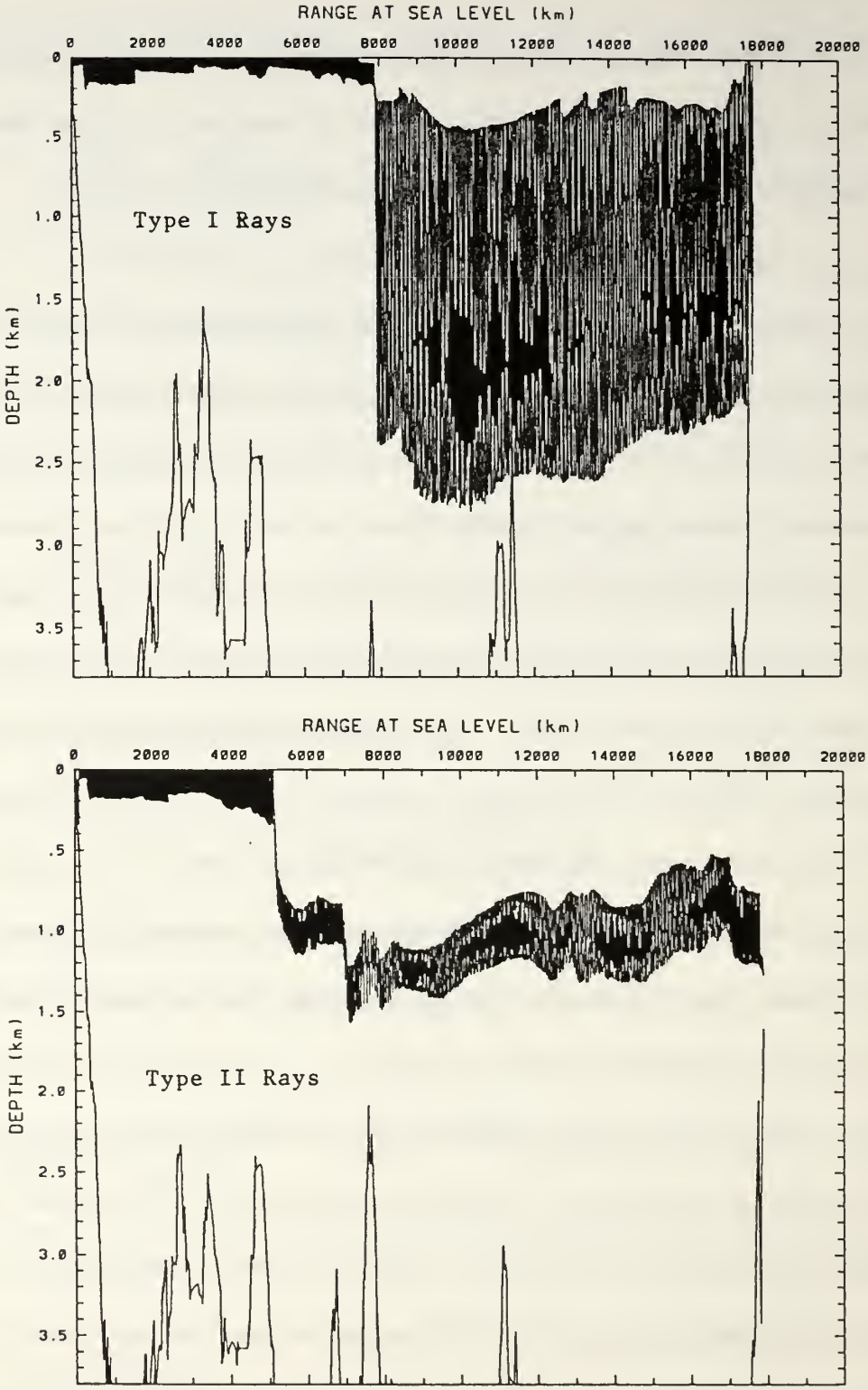

Figure 2.4: Vertical Distribution of Type I and Type II Acoustic Ray Energy.

Type 2 rays are also trapped near the surface in the half channel regime, but less so than the Type 1 rays. At a range of about 5000 km, the Type 2 rays encounter a stronger vertical gradient which refracts the acoustic energy downward toward the deep sound channel. Also at about 5000 km, the deep sound channel axis gradually increases depth to about 1000 m over a distance of about 600 km. This gradual entry angle into the deep sound channel causes the Type 2 rays to have a much smaller vertical fluctuations about the axis.

Since the Type 1 rays oscillate further from the minimum sound speed at the channel axis, they traverse water with a higher sound speed during most of their path. They can thus be expected to arrive sooner than the axially trapped Type 2 rays. In addition, note that the upper portion of the Type 1 ray trajectory lies within the oceanic thermocline where sound speed variability is the greatest. Therefore, although the Type 1 rays are more representative of the upper ocean (which is more closely coupled to the atmospheric temperature), they will also exhibit more variability in arrival time.

The travel time for a specific ray χ_i (the i^{th} ray path) can be found by integrating the sound slowness (inverse speed) over the path

$$\tau_i = \int_{\chi_i} \frac{ds}{c} \tag{2.14}$$

The perturbations in sound speed can then be viewed as deviations from some arbitrary reference speed c_0,

$$c = c_0 + \delta c \ , \tag{2.15}$$

so that the total travel time becomes a constant travel time with a perturbation

$$\tau_{i_0} + \delta\tau_i = \int_{P_i} \frac{ds}{c_0 + \delta c} . \tag{2.16}$$

Since for our purposes, δc is much smaller than c_0, an approximation from the binomial expansion can be used

$$\tau_{i_0} + \delta\tau_i = \int_{P_i} \frac{1}{c_0} \frac{ds}{\left(1 + \frac{\delta c}{c_0}\right)} = \int_{P_i} \frac{1}{c_0}\left(1 - \frac{\delta c}{c_0}\right) ds = \int_{P_i}\left(\frac{1}{c_0} - \frac{\delta c}{c_0^2}\right) ds \tag{2.17}$$

The travel time perturbation is then given as

$$\delta\tau_i = -\int_{P_i} \frac{\delta c}{c_0^2} ds \tag{2.18}$$

2. Propagation Losses

Little prior experience exists with propagation losses over the 18 Megameter ranges traversed in this experiment. However expanding the current theories to global ranges can lend some insight to the order of magnitude losses that can be expected. The most significant energy loss should be spreading losses due the expanding area of the phase surface as the wave propagates from the source. A secondary loss due to absorption and scattering can also be expected.

a. Spreading Losses

A worst case scenario involves totally lossy boundaries which can be approximated by spherical spreading in an iso-velocity fluid (Birdsall, 1990). Invoking energy conservation in a lossless medium on an acoustic phase surface propagating outward from a point source gives the spherical wave solution at range **r** from the source

$$p = \frac{p_0}{4\pi r^2} e^{j(\omega t - \vec{k}\vec{r})} \quad (2.19)$$

It follows that the energy contained on a sphere of unit diameter about the point source, p_0, is conserved on the expanding phase surface as shown by

$$p_0 = 4\pi r_1^2 I_1 = 4\pi r_2^2 I_2 \quad (2.20)$$

where I_1 and I_2 are the phase surface intensities at ranges 1 and 2 respectively. The transmission loss is then given by

$$TL = 10\log\frac{I_1}{I_2} = 10\log r_2^2 = 20\log r_2 \quad (2.21)$$

which yields a loss of 145.1 dB over 18 Mm.

A best case scenario assumes totally reflective boundaries which can be approximated by a flat earth similar to a right cylinder with an 18 Mm radius and 5 km height (Birdsall, 1990). The cylindrical wave solution is

$$p = \frac{p_0}{4\pi r} e^{j(\omega t - \vec{k}\vec{r})} \quad (2.22)$$

The transmission loss is then given by 10 times the log of the intensity on the cylindrical phase surface relative to a unit sphere

$$TL = 10\log\left(\frac{2\pi r_2 H}{4\pi r_1^2}\right) = 10\log\left(\frac{r_2 H}{2}\right) \quad (2.23)$$

where r_1 is 1 m and H is the depth of water in meters. This formula yields a transmission loss of 106.5 dB over 18 Mm. This flat earth model assumed 2π x 18 Mm = 113.1 Mm as the cylindrical circumference although the circumference is physically constrained to the 40 Mm circumference of the earth. Birdsall suggests using a

convergence factor of 10 log(113.1/40) = 4.5 dB to correct for this difference which reduces the predicted transmission loss to 102 dB. If the depth is constrained to a nominal 1000 m height of the sound channel, the predicted transmission loss is reduced to 99.5 dB. Both models are pessimistic in that some of the observed intensities will be greater than the average predicted intensity. However, both models are also optimistic in that they assume no interaction with the surface, the bottom, or with oceanic fronts.

b. Absorption Losses

Absorption at 57 Hz is usually considered negligible, but over 18 Mm the cumulative loss becomes significant. The 57 Hz carrier frequency falls below the peak relaxation frequencies of both $MgSO_4$ and $B(OH)_3$ but is dominated by the $B(OH)_3$ relaxation process (Clay and Medwin, 1977). For a depth of about 1000 m and a temperature of 4° C, we can find an absorption coefficient α from (Thorp, 1967)

$$\alpha = \frac{0.1f^2}{1 + f^2} + \frac{40f^2}{4.1 + f^2} + 0.000275f^2 + 0.0007 \qquad (2.24)$$

where **f** is the carrier frequency and α is in dB per kyd. The terms in the above equation are due to boric acid ionization, magnesium sulphate relaxation, viscosity shear, and low frequency scattering respectively. The last term is dominant at 57 Hz and is taken from Figure 2.5 which compares the empirical values of low-frequency attenuation to the extrapolated relaxation absorption. An absorption coefficient of 0.0007 dB per kyd or 0.758 dB per Mm gives an absorption loss of 13.6 dB over 18 Mm.

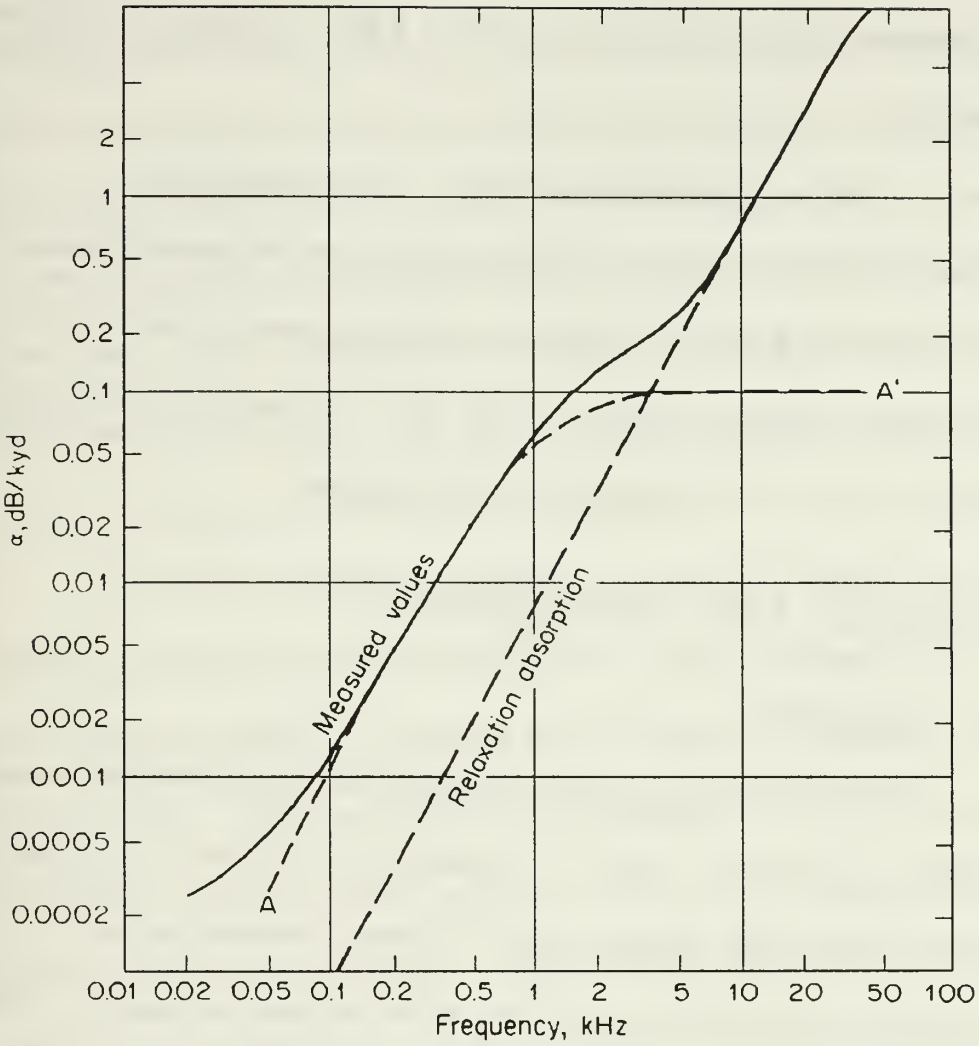

Figure 2.5: Low Frequency Absorption Coefficient [Ref. 15].

3. Signal-to Noise Ratio

The signal level at the receiver hydrophones can be predicted by

$$\textit{Received Signal Level} = \textit{Source Level} - \textit{Transmission Loss} \qquad (2.25)$$

where all levels are interpreted as band levels. This gives levels between 54.3 and 97.4 dB rel 1 μPa at the receiver hydrophones depending on the spreading model used.

The predicted 57 Hz noise at the receiver, shown in Figure 2.1, is primarily due to shipping and falls between 75 to 93 dB rel 1 μPa. The signal-to-noise level can be predicted by

$$SNR = Receiver\ Signal\ Level - Receiver\ Noise\ Level \tag{2.26}$$

which gives an SNR from -38.07 to 22.4 dB depending on the assumed scenario. Using these values, between 0 and 56.93 dB of array and/or processing gain is needed to meet our arrival time resolution criteria of 18.86 dB. The method of maximizing the processing gain will be described in the following section.

C. MAXIMAL LENGTH SEQUENCES

1. Introduction

Impulse excitation (a signal of infinitesimal length and infinite magnitude) is a convenient and powerful method for measuring the response of a system and for determining the travel time through a media. The oceanic acoustic medium can be treated as a large, time-varying distortionless filter whose response to impulsive excitation is simply the sum of delayed impulses traveling along the individual paths (Dees, 1989)

$$h(t) = \sum_{i=1}^{Paths} a_i \, \delta(t - \tau_i) \tag{2.27}$$

where $\mathbf{a_i}$ is the amplitude of the $\mathbf{i^{th}}$ path, and τ_i is the travel time along the $\mathbf{i^{th}}$ path.

Unfortunately, ideal impulse transmissions are physically difficult to achieve. The amplitude and length of a pulse are limited by the peak power and bandwidth, respectively, of the transmitter. Impulsive acoustic signals can be generated by explosive

or implosive sources but these have an uneven frequency distribution of energy, and are generally not reproducible. Another solution is to use pseudorandom noise to provide a means of pulse compression. In this method, a long coded signal is transmitted whose period, frequency distribution, and energy are deterministic and can be tailored to meet experimental requirements. The received signal can then be cross-correlated, through a matched filter, with the original code to produce a single pulse with a much higher amplitude and shorter length than the transmitted signal. In addition, the coded periodic signal can be repeated for additional processing gain.

The pulse compression technique used for the Heard Island Feasibility Experiment was to utilize a phase-modulated carrier signal to transmit a specific sequence of ones and zeros known as a maximal length sequence or m-sequence (Ziemer and Peterson, 1985). The most important characteristic of the m-sequence is its autocorrelation, which is constant except at a shift of zero (corresponding to perfect correlation with the transmitted code), making the sequence analogous to white noise. The width of the correlated pulse is equal to the width of one individual digit of the code, T_c, which was 0.087 seconds for this experiment. Details of the specific signal used in this experiment are given in Chapter III.

The amplitude at the correlated zero-shift digit has an increase over amplitude of the coded signal equal to the number of digits in the code, N. The ideal pulse compression gain for a single sequence of code is given by **10 log(N)**. Processing gain is thus dependent on the code length which is in turn limited by the length of time the ocean can be assumed stationary. One of the goals of this experiment was to determine

the maximum effective sequence length. This experiment used sequence lengths of 255, 511, 1023, and 2047 digits, corresponding to 22.37, 44.8, 89.7, and 179.6 seconds respectively. The ideal processing gain for these sequences was then 24.06, 27.08, 30.1, and 33.11 dB respectively. In addition, if the m-sequence code is repeated **L** times, a gain of **10 log(L)** can be added to the single sequence gain under ideal conditions. The amount of correlated repetition is dependent on the stability of the arrival structure which is in turn influenced by the meso, gyre, and basin scale variability of the ocean environment.

2. M-Sequence Generation

M-Sequences are generated using a simple binary shift register whose structure is defined by a specified polynomial (Ziemer and Peterson, 1985). A general shift register generator is shown in Figure 2.6. A deterministic and periodic sequence of bits can be output from any one of the registers, with period NT_c. Although an equally valid m-sequence is generated in both the forward and reverse order, this experiment

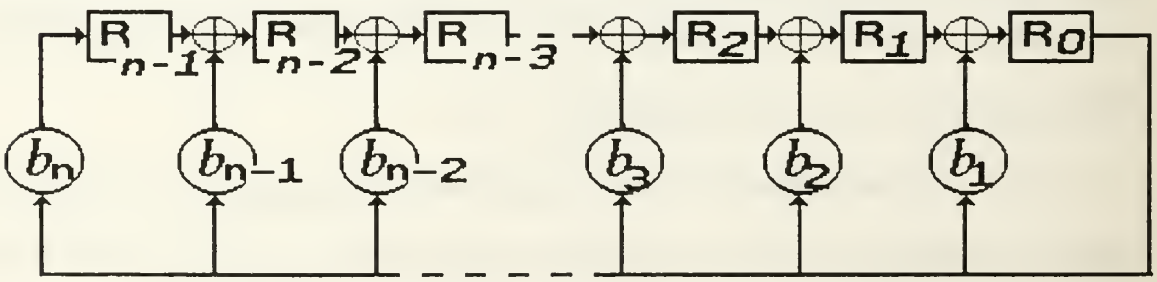

Figure 2.6: General sequence shift register generator of degree n.

consistently used the sequential output from the 0^{th} register, $\mathbf{R_0}$, such that the consecutive bits generated at the output comprised the m-sequence. In addition, although the initial state of the shift register is arbitrary, this experiment consistently used an initial sequence of all zeroes followed by a one.

The shift register structure is specified by a primitive polynomial such as

$$g_D = b_n D^n + b_{n-1} D^{n-1} + \ldots + b_1 D + b_0 \qquad (2.28)$$

where **n** is the number of delays in (or degree of) the shift register generator, **D** is the unit delay and $\mathbf{b_n}$ are the feedback weighting coefficients. The weighting coefficients take on values of one or zero indicating connection or no connection, respectively, to the $\mathbf{n^{th}}$ register. Table 8-5 from Ziemer and Peterson lists octal representations (referred to as the octal law) of the binary coefficients corresponding to particular generating polynomials. M-sequences do not repeat a register state until after $\mathbf{2^n-1}$ delays. The governing polynomial is known as a primitive polynomial since, for a specified number of shift register stages, a sequence of maximal-length period is produced before the sequence repeats. The signal specifications for the m-sequence code used in this experiment are detailed in Chapter III. In practice, the m-sequence digits are transformed by mapping **[1,0]** to **[-1,1]** which enables use of faster correlation procedures. Because many people are familiar with binary mathematics, however, all arithmetic operations for polynomials in this thesis are performed using finite-field arithmetic (modulo two).

3. Characteristics

The output of an **n**-stage m-sequence shift register generator with digit length T_c and length $N = 2^n - 1$ can be represented as

$$c = \sum_{i=-\infty}^{\infty} a_i U_{t-iT_c} \tag{2.29}$$

where $a_i = (-1)^{b_i}$, and **U** is a unit pulse beginning at 0 and ending at T_c. Its discrete two-sided autocorrelation function R_k for **k** cyclic shifts of the generator is given by

$$R_k = \begin{bmatrix} 1.0 & for \quad k=mN \\ -\frac{1}{N} & for \quad k \neq mN \end{bmatrix} \tag{2.30}$$

where **m** is any integer.

The non-discrete autocorrelation function of the m-sequence code for time delay of τ is given by

$$R_c = \left(1 - \frac{\tau_\epsilon}{T_c}\right) R_k + \left(\frac{\tau_\epsilon}{T_c}\right) R_{k+1} \tag{2.31}$$

where $\tau_\epsilon = \tau - (N-1)T_c$ (Ziemer and Peterson, 1985)

If $0 \leq \tau \leq T_c$, then $k = 0$ and $\tau_\epsilon = \tau$ so that

$$\begin{aligned} R_c &= \left(1 - \frac{\tau}{T_c}\right) - \frac{1}{N}\left(\frac{\tau}{T_c}\right) \\ &= 1 - \frac{\tau}{T_c}\left(1 + \frac{1}{N}\right) \end{aligned} \tag{2.32}$$

If $T_c \leq \tau \leq (N-1)T_c$, then $k \neq mN$ and $(k+1) \neq mN$ for any integer **m** so

$$\begin{aligned} R_c &= \left(1 - \frac{\tau_\epsilon}{T_c}\right) - \frac{1}{N} + \frac{\tau_\epsilon}{T_c} - \frac{1}{N} \\ &= \frac{-1}{N} \end{aligned} \tag{2.33}$$

If $(N-1)T_c < \tau < (NT_c)$, then $k = N - 1$ and $k + 1 = 1$ so that

$$R_c = \left(1 - \frac{\tau_e}{T_c}\right) - \frac{1}{N} + \frac{\tau_e}{T_c}$$
$$= \frac{\tau_e}{T_c}\left(1 + \frac{1}{N}\right) - \frac{1}{N} \qquad (2.34)$$

Now Eqs. 32, 33, and 34 define one complete cycle of the maximal length sequence autocorrelation which is illustrated in Figure 2.7.

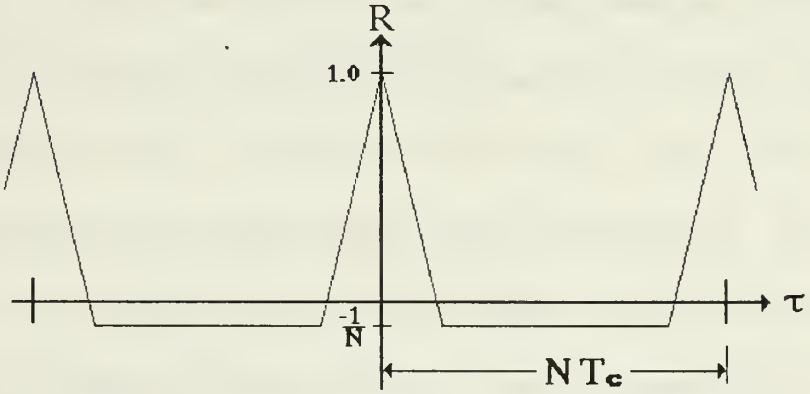

Figure 2.7: Autocorrelation function for a m-sequence of length N.

The Wiener-Khinchine theorem shows that the Fourier transform of an autocorrelation function describes the frequency distribution of power and is referred to as the power spectral density or power spectrum of the time series (Bendat and Piersol, 1986). The power spectrum of the baseband m-sequence is then

$$S_c = \sum_{i=-\infty}^{\infty} A_i \, \delta(f - i f_N) \qquad (2.35)$$

where $A_0 = 1/N^2$, $A_i = [(N+1)/N^2]\ \mathrm{sinc}^2(i/N)$, and $f_N = 1/NT_c$. This spectrum appears as a series of discrete impulse lines separated by the code repetition frequency $1/NT_c$ with the line amplitude envelope defined by

$$A = \left(\frac{N+1}{N^2}\right) sinc^2(f T_c) \tag{2.36}$$

for all except the DC line which has an amplitude of $1/N^2$.

If the phase shift angle θ is chosen such that

$$\theta = \tan^{-1}\left(\sqrt{N}\right) \tag{2.37}$$

then the spectrum of the carrier signal will fall exactly on the envelope of impulses and result in the maximum signal-to-noise ratio after demodulation and sequence removal (Spindel, 1985). The optimum phase shifts for pulse compression are then approximately 86.42°, 87.47°, 88.21°, and 88.73° for the 255, 511, 1023, and 2047 digit sequences respectively. The transmitted signal for this experiment used a 90° phase change between digits for all modulations which provided roughly half the power in the carrier. This phase shift was chosen because it produces about 6 dB steps in the power spectrum of the pentaline signal which facilitates graphical estimation of the signal-to-noise ratio as discussed in the following Chapter.

When the m-sequence c is used to biphase modulate a carrier wave having power P_c and frequency f_0 the resultant signal is given by

$$s = \sqrt{2 P_c}\; c \cos(2\pi f_0 t) \tag{2.38}$$

The power spectrum of the modulated carrier signal S_s is then the convolution of the carrier power spectrum S_c and the m-sequence power spectrum

$$S_s = \left[S_c * \frac{A}{2}\delta(f - f_0)\right] + \left[S_c * \frac{A}{2}\delta(f + f_0)\right] \tag{2.39}$$

and the effective result is a translation of the discrete spectrum S_c upward and downward by a frequency f_0. The spectrum of the transmitted signal is then a band of impulses

centered at f_0 and separated by $1/NT_c$. Figure 2.8 illustrates the amplitude envelope of the individual impulses from the transmitted signal.

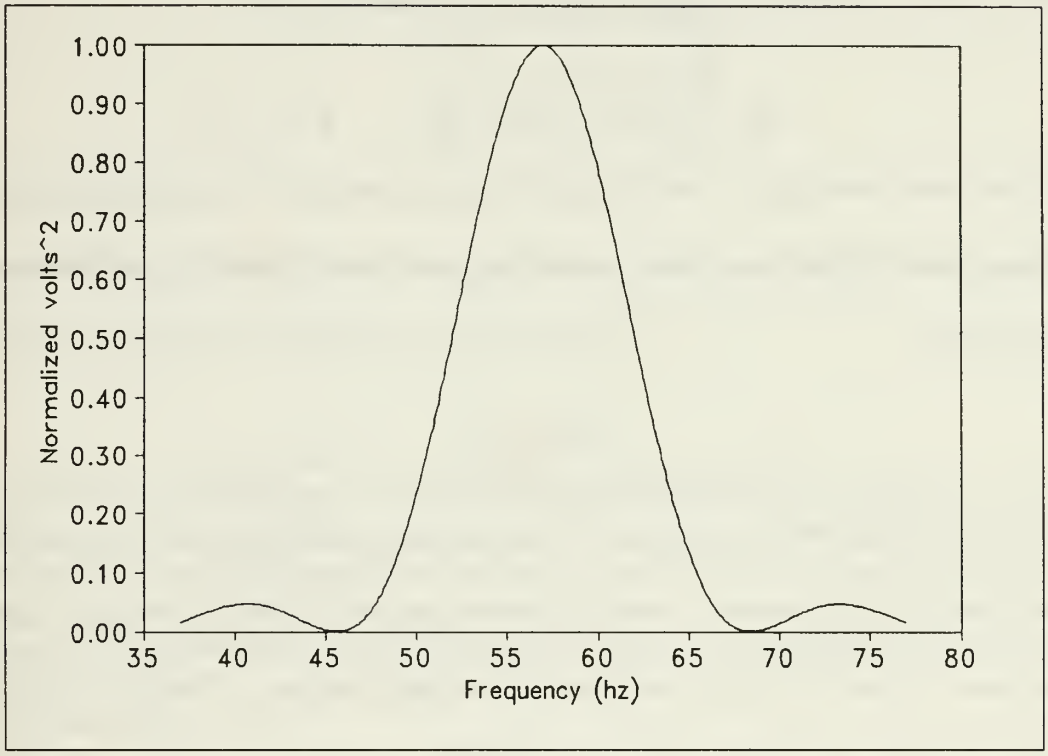

Figure 2.8: Envelope of Power Spectrum impulses in the Transmitted M-Sequence Signal.

4. Autocorrelation Process

Maximal length sequences have several properties which make it attractive as a pulse compression code. Its autocorrelation function has a large magnitude and short duration which enhances arrival time estimates. It is also deterministic in that once the polynomial is specified, all output parameters are known. Additionally, it produces a

repeatable, periodic signal which can be easily implemented. The single major drawback is that standard autocorrelation procedures are computationally intensive.

The sample autocorrelation function $\mathbf{R_{xx}}$ of a stationary (ergodic) random process with a zero mean can be estimated as (Spindel, 1985)

$$\hat{R}_{xx} = \frac{1}{t - \tau} \int_0^{t-\tau} x_t x_{t+\tau} \, dt \qquad 0 \le \tau \le t \tag{2.40}$$

for time **t** and time delay τ. For N data values $\{x_n\}$, **n=1,2,...N**, sampled at equally spaced time intervals δ**t** from a stationary record $x_{(t)}$ with a zero mean, $\mathbf{R_{xx}}$ can be numerically computed by

$$\hat{R}_{xx} = \frac{1}{N - \Gamma} \sum_{i=1}^{N-\Gamma} x_i x_{i+\Gamma} \qquad \Gamma = 0,1,2,\ldots,m \tag{2.41}$$

where Γ is the lag number and **m** is the maximum lag number (**m** < **N**). Since the arrival time is not known in advance, standard correlation methods require the input signal to be correlated with all N shifted versions of the original sequence. Thus N^2 multiplications are necessary for computing a standard FFT based cross-correlation. The following paragraphs describe a less computationally intensive autocorrelation method which takes advantage of the m-sequence structure.

Using a 3 register generator output as an example with an octal law of 13 and a polynomial of degree 3, the forward signal code S is

$$S = [\ 1\ 0\ 0\ 1\ 1\ 1\ 0\ ] \tag{2.42}$$

To correlate S with all seven shifted versions of itself, we can construct a matrix **M** of all possible versions

$$M = \begin{bmatrix} 1 & 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 & 1 \end{bmatrix} \tag{2.43}$$

When the digits of S and **M** are mapped [1,0] to [-1,1], multiplying the signal by the correlation matrix will result in the autocorrelation,

$$R_{sm} = M S \tag{2.44}$$

This product is the entire goal of the initial signal processing but requires a computer intensive N^2 add-multiply operations. A faster, more efficient algorithm, called the Fast Hadamard Transform can be used to reduce the number of operations required to perform the autocorrelation.

5. The Hadamard Matrix

To facilitate discussion of the Fast Hadamard Transform, it is necessary to introduce the Hadamard matrix (Ziemer and Peterson, 1985). The Sylvester-type Hadamard matrix has a recursive form for higher orders given by

$$H_1 = [1] , \qquad H_2 = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} , \qquad H_{i+1} = \begin{bmatrix} H_i & H_i \\ H_i & -H_i \end{bmatrix} \tag{2.45}$$

For our third order example, the matrix **H** is

$$H = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 & 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 & 1 & -1 & -1 & 1 \\ 1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 \\ 1 & -1 & 1 & -1 & -1 & 1 & -1 & 1 \\ 1 & 1 & -1 & -1 & -1 & -1 & 1 & 1 \\ 1 & -1 & -1 & 1 & -1 & 1 & 1 & -1 \end{bmatrix} \quad \textbf{(2.46)}$$

It is useful to map the Hadamard matrix from [1,-1] to [0,1] as follows:

$$H = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 & 0 & 1 \end{bmatrix} \quad \textbf{(2.47)}$$

H can be factored into a binary counting (from 1 to 7 in this example) matrix **A** and its transpose $\mathbf{A}^{\mathbf{T}}$, which is also a binary counting matrix as given by

$$H = AA^T = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 0 & 0 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \\ 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{bmatrix} \tag{2.48}$$

The correlation matrix **M** can be similarly factored into two matrices **B** and **C** which contain all binary combinations, but not as simply. The first matrix **B** is formed from the sequential output of the shift register, but bit reversed (from right to left) and in the reverse order (from bottom to top). The original order is then preserved by shifting the rows of the matrix to bring the 3x3 identity matrix to the top. For our third order m-sequence example, the factor matrix **B** is given by

$$B = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 0 & 1 \end{bmatrix} \tag{2.49}$$

The second matrix **C** is formed from any 3 shifted versions of the m-sequence, ie

$$C = \begin{bmatrix} 1 & 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 0 & 0 & 1 & 1 \end{bmatrix} \tag{2.50}$$

It is easily verified that

$$BC = M \tag{2.51}$$

By adding a leading column of zeros to **B**, forming **B'**, and adding a leading row of zeros to **C**, forming **C'**, a new matrix **M'** can be formed by

$$M' = B'C' \tag{2.52}$$

Note that **M'** contains all possible combinations of ones and zeros, as does **A**, albeit in a different order. If mapping matrices can be found such that **QA = B'** and $\mathbf{A^TP = C'}$, then the same matrices will map the Hadamard matrix to the m-sequence matrix

$$M' = B'C' = QAA^TP = QHP \tag{2.53}$$

6. The Fast Hadamard Transform

Recall that correlation of the received signal **S** with the m-sequence code **M** was performed by the matrix product of **S** and **M**. By adding a leading zero to **S**, forming **S'**, equation (2.44) becomes

$$R'_{sm} = M'S' = QHPS' \tag{2.54}$$

Equation (2.54) shows that matrix **M** is not needed and the Hadamard matrix can be used in its place. An efficient method of forming the matrix product of **QHPS'** is the Fast Hadamard Transform (FHT). The FHT takes advantage of the fact that any vector **PS'** which is multiplied by the unmapped Hadamard matrix (consisting of +1's and -1's), results in a sum of all the terms of **PS'** with various + and - weighting. This summation can be effectively utilized to cancel random noise amplitudes where a random sum of positive and negative values will equal 0. In addition, non-random amplitudes contained in the m-sequence code will sum coherently to a value equal to the sequence length times

the absolute amplitude of a single value. For our third degree example an arbitrary vector given by

$$PS' = \begin{bmatrix} a \\ b \\ c \\ d \\ e \\ f \\ g \\ h \end{bmatrix} \tag{2.55}$$

after multiplication by the Hadamard matrix becomes

$$HPS' = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 & 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 & 1 & -1 & -1 & 1 \\ 1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 \\ 1 & -1 & 1 & -1 & -1 & 1 & -1 & 1 \\ 1 & 1 & -1 & -1 & -1 & -1 & 1 & 1 \\ 1 & -1 & -1 & 1 & -1 & 1 & 1 & -1 \end{bmatrix} \begin{bmatrix} a \\ b \\ c \\ d \\ e \\ f \\ g \\ h \end{bmatrix} = \begin{bmatrix} a+b+c+d+e+f+g+h \\ a-b+c-d+e-f+g-h \\ a+b-c-d+e+f-g-h \\ a-b-c+d+e-f-g+h \\ a+b+c+d-e-f-g-h \\ a-b+c-d-e+f-g+h \\ a+b-c-d-e-f+g+h \\ a-b-c+d-e+f+g-h \end{bmatrix} \tag{2.56}$$

which, when permuted by **Q**, is the autocorrelation function $\mathbf{R}_{xx}$. This technique is called the Fast Hadamard Transform. When written in the form of a flow graph as in Figure 2.9, the product **HPS'** has the same form as the well known fast Fourier transform, where the complex multiplications are set to one (Cohn and others, 1976 and Borish and others, 1983). Note that there is no bit reversal or multiplication by a phase factor as is required by the Fast Fourier Transform. Since the multiplication by **P** and **Q** has been replaced

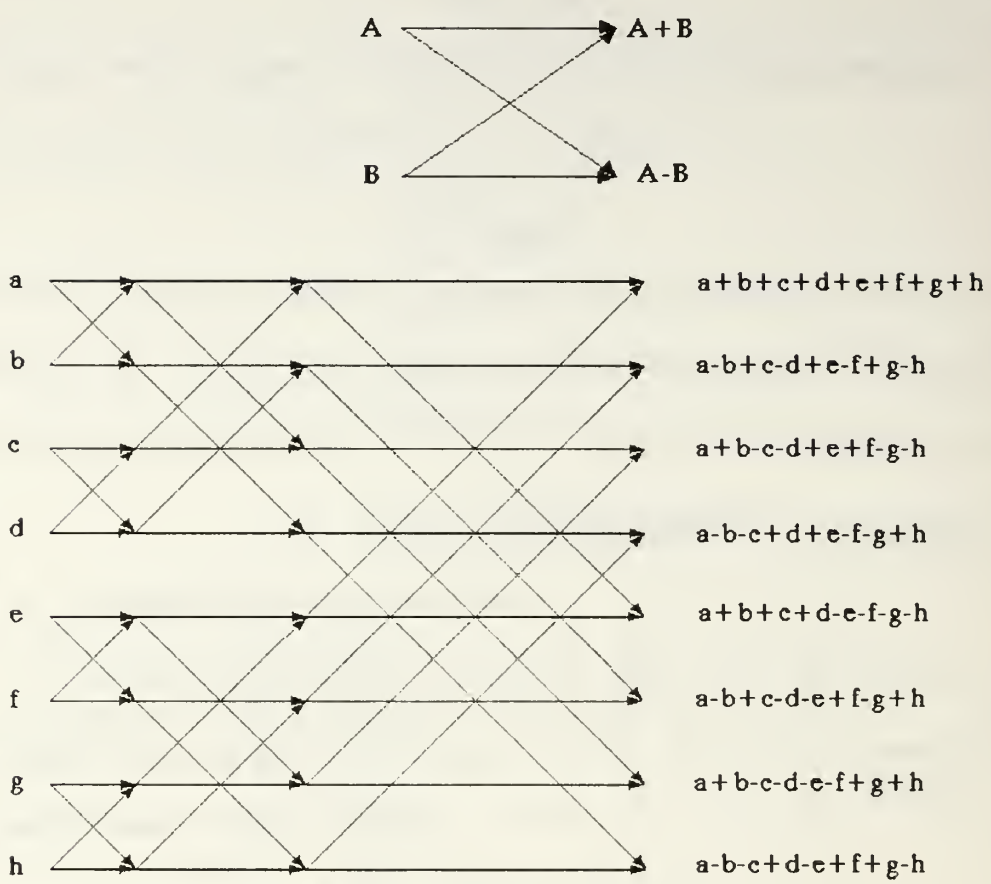

Figure 2.9: The Basic Fast Hadamard Transform Element for Cascading Additions and the Full Diagram for an 8 Point FHT.

by permuting the matrix positions, there is no multiplication required. The increase in speed of execution is difficult to compare with the FFT since this method only requires additions. During the correlation procedure, **a=0** so no new information is added. The

zeroth position result can be seen to be the sum of all the elements of the code and is thus equal to the DC pedestal. This pedestal can be removed by subtracting the zeroth position from all other positions. All that remains is to determine the permutation matrices **P** and **Q**.

7. Vector Order Permutation

The permutation matrices **P** and **Q** must be found such that $\mathbf{QA} = \mathbf{B'}$ and $\mathbf{A^TP} = \mathbf{C'}$. Matrices **A**, $\mathbf{A^T}$, **B'**, and **C'** contain all possible arrangements of ones and zeros in 3 digit rows or columns. A natural index for each row or column is then its equivalent octal value. The indices are important in that they allow 'multiplication' by the permutation matrices by rearranging the order of the signal vector, rather than direct multiplication. Note that no multiplications are required, only the reordering. For a given code, the permutations can be evaluated once, prior to signal processing, and the result stored as an index array to be applied to each vector.

8. Correlation Summary

The following is a summary of the autocorrelation procedure:

1. Augment the signal vector S with a zero in the zeroth position to form S'.
2. Permute the augmented input vector S' according to **P**.
3. Perform the Fast Hadamard Transform additions as indicated by the flow graph in Figure 2.9.
4. Permute the vector resulting from the FHT according to **Q**.
5. Remove the zeroth entry and subtract it from all other elements to remove the DC bias.

III. SIGNAL PROCESSING

A. SIGNAL SPECIFICATION

1. Continuous Wave

The CW signal was a pure sinusoidal waveform transmitted at a nominal array power level of 213 dB rel 1uPa@1m. This signal was useful in estimating the frequency stability of the arrivals, and in comparing signal levels to noise in the frequency band of interest. However, the extreme power levels at which the transducers were driven coupled with severe weather conditions in the vicinity of Heard Island caused variability in the output level performance of the transmitter.

2. 3/10 Pentaline

The 3/10 pentaline signal has five major spectral lines hence the name pentaline. It consists of 3 digits of 10 carrier cycles each at 57.0 Hz, a total of 30 cycles per period, repeated continuously for one hour. This produces a nominal bandwidth of 57.0 Hz / 10 cycles per digit = 5.7 Hz. The signal is specified by

$$s = A\cos(2\pi f_c t + b_n \theta) , \tag{3.1}$$

where b_n = ||: -1, +1, +1 : ||, corresponding to law 7_{octal} or digits of 1,0,0 in binary and n is incremented once every 10 carrier cycles. The first digit is phase shifted by -45 degrees, the second and third are phase shifted by +45 degrees. There are 57.0 Hz / 30 cycles per period = 1.9 periods per second, which produces a spectral line separation of

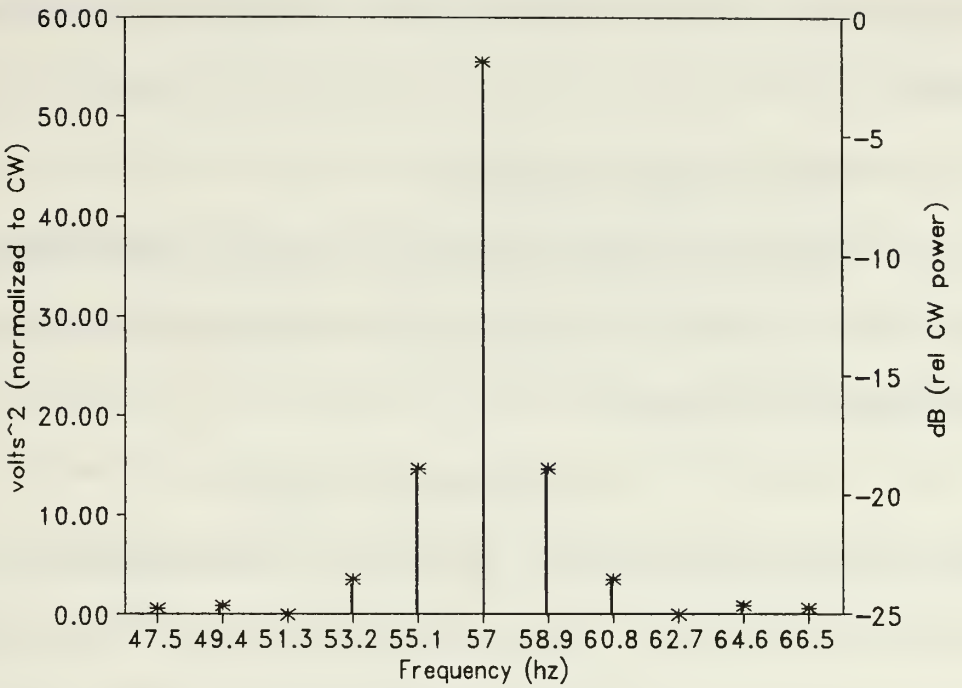

Figure 3.1: Power Spectrum of the Pentaline Signal.

1.9 Hz. Figure 3.1 illustrates the power distribution in the frequency domain. The power levels rel 57.0 Hz CW signal are tabulated below.

LOWER FREQ. (Hz)	UPPER FREQ. (Hz)	POWER (Percent of CW)	POWER (dB rel CW)
57.0	57.0	55.5	-2.5
55.1	58.9	14.7	-8.3
53.2	60.8	3.5	-14.5
51.3	62.7	0.0	-∞
49.4	64.6	0.8	-20.8
47.5	66.5	0.5	-22.9

Note that the power decreases in approximately 6 dB steps in line amplitude which permits graphical estimation of received signal-to-noise ratios from the observed power spectrum.

3. Maximal Length Sequences

The maximal length sequences each consist of a series of digits of 5 cycles each at 57.0 Hz, producing a digit duration of 87.7 milliseconds and a bandwidth of 11.4 Hz. The signal is specified as

$$s = A\cos(2\pi f_c t + b_n \theta) , \tag{3.2}$$

where **n** is incremented once every 5 carrier cycles. Phase modulation is +/- 45 degrees as it is for the 3/10 pentaline. The initial digits are 0,0,...0,0,1 i.e., the maximum string of 0's followed by a 1. The sequence lengths are 255, 511, 1023, or 2047 digits with their characteristics tabulated below.

Digits	LAW (octal)	PERIOD (seconds)	Periods/Hr
255	537	22.36	160.9
511	1473	44.82	80.3
1023	2033	89.82	40.0
2047	5747	179.56	20.0

B. DATA COLLECTION AND PRE-PROCESSING

1. Amplification

The receiving array of 32 hydrophones continuously collected 32 channels of data measuring the acoustic field at depths between approximately 345 and 1740 meters.

The hydrophones had a nominal sensitivity of -170 dB (rel 1V/μPa) corresponding to 316 μPa/μV. Therefore, the data can easily be converted directly to pressure expressed in micropascals. Received hydrophone data underwent a linear fixed gain of 20 dB at a patch panel and then was linearly preamplified for a gain of 6 dB before processing through a variable gain amplifier. The variable gain amplifier automatically switched gain (1, 8, 84, 512, or 4096) depending on the input level to optimize the 10 volt dynamic range of the analog to digital converter and thus the precision of the recorded signal data. All data analyzed in this thesis has been normalized to the hydrophone output voltage levels using the program convert.c (listed in Appendix B) which is an implementation of the program normize.c written by Von der Heydt of Woods Hole Oceanographic Institution (per E-mail communication dtd 17 April 1991).

2. Analog Filtering

As noted above, the maximum bandwidth for the transmissions is 11.4 Hz for all of the m-sequence signals. It is desirable to bandpass filter the received data to eliminate any out-of-band noise. Therefore, in addition to amplification, the signal underwent bandpass filtering prior to recording. The highpass filter had a -6 dB point at 10 Hz with a rolloff of -30 dB/octave. The lowpass filter had a -3 dB point at 80 Hz with a rolloff of -48 dB/octave.

3. Analog-to-Digital Conversion

Figure 3.2 illustrates the data collection and pre-processing apparatus aboard *R/V Point Sur*. The acoustic signal was digitally sampled at 228 Hz for transmissions of

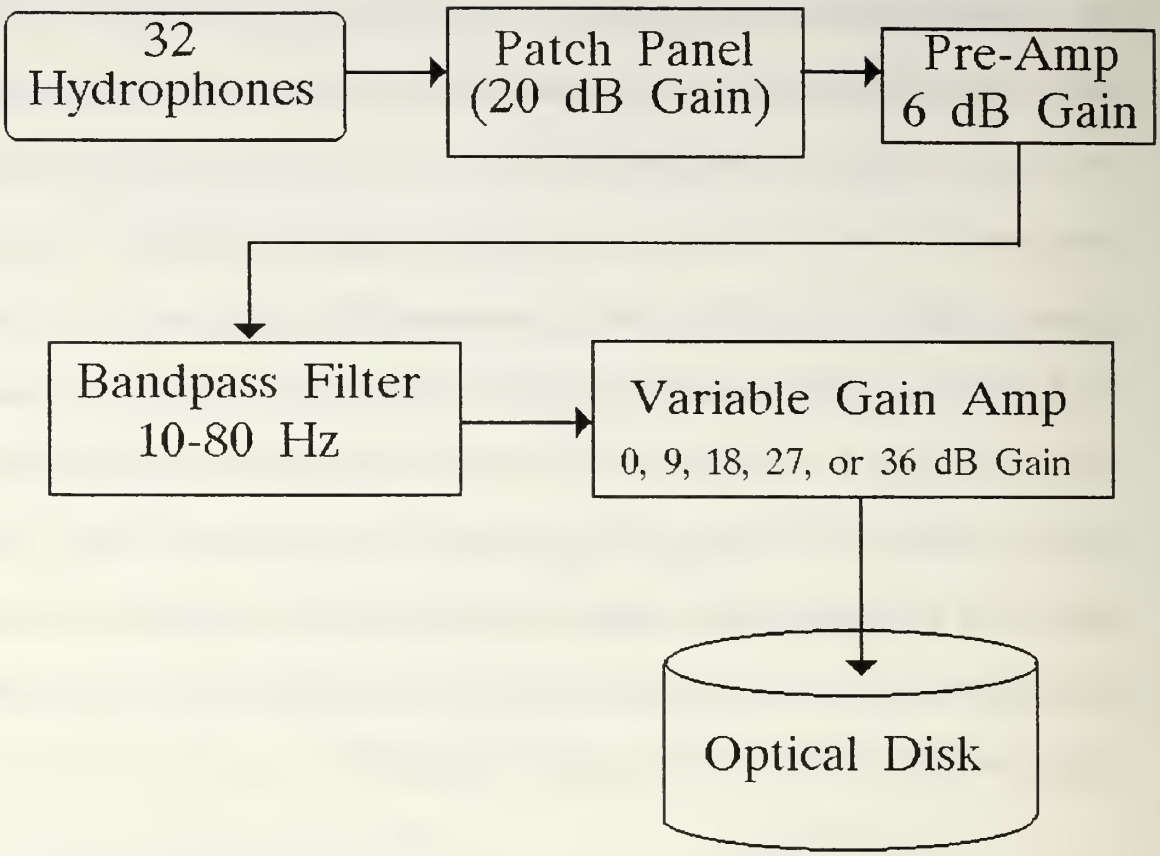

Figure 3.2 Data Preprocessing

26 January 1991 to January 1991 and then at 456 Hz from January to February 1991. These sampling frequencies are 4 and 8 times the Nyquist frequency respectively and are an integer multiple of the optimum carrier frequency f_c. Sampling in this manner simplifies processing and interleaving of data, since it produces an integer number of data points per digit. The data was digitally recorded continuously during the 1 hour reception intervals onto both 1 Gigabyte optical-disk and VHS tape media. After normalization to hydrophone output levels, the data was stored in an array of binary single precision floating point values. Real time processing and signal monitoring was conducted using

both a Concurrent 6400 Real Time Unix processor and more conventional spectrum analyzers.

4. Clipping

During postprocessing of the data, occasional broadband spikes of high amplitude noise of undetermined origin were noticed which lasted approximately one second. These noise transients tended to saturate the local time series to a degree that attempts to recover the signal near the spike were futile. Since these spikes in the time domain data had no relation to the transmitted signal but significantly raised the background noise level, they can be clipped without significantly degrading the received signal. During preprocessing, if the absolute value of a data point exceeded twice the standard deviation from the mean of its local 10 second buffer it was clipped or limited to that value. By clipping to a standard deviation from a localized mean, short term noise anomalies are eliminated but the general characteristics of signal-to-noise in the slowly varying ambient noise field are retained. Clipping was incorporated into the resampling module but occurred prior to resampling.

5. Doppler Correction

Both the transmitting and receiving ships and arrays were acted upon by the forces of tides, currents, winds, etc., which combined to produce some amount of relative drift between them. In addition, the source array was towed during transmission periods as required to maintain steerage of the source vessel with minimum thrust. Although both vessels were operated to minimize array motion (especially accelerations), some relative

movement was unavoidable and because of this the received signal center frequency was often Doppler shifted away from the transmitted center frequency. Since the motions were slow and generally uniform, it is sufficient to make corrections for first order Doppler shifts and neglect acceleration effects (Birdsall and others, 1987, 1988).

When the signal s incurs a Doppler frequency shift due to relative velocity, v, the received signal experiences an expansion or compression in time. The Doppler shifted signal s_d can be expressed as

$$S_d = S\left[\left(1 + \frac{v}{c}\right)t - \tau_i\right] \tag{3.3}$$

where c is the average phase speed of sound in water given in meters per second, and τ_i is the delay of the i^{th} arrival from the transmitter to the receiver. Since the objective of the Heard Island Feasibility Experiment was to examine the resolution of the arrival structure and not to determine the travel time itself, τ_i is arbitrary and only important in its relation to other arrival delays. Therefore it will be assigned a reference value of zero for this discussion. The magnitude of the Doppler shifted power spectrum is given by

$$\left|S_d\right| = \left|S\left[\frac{f}{\left(1 + \frac{v}{c}\right)}\right]\right| \tag{3.4}$$

When the Doppler shifted signal is digitally sampled at a frequency of f_s the value of the n^{th} sample is given by

$$S_{dn} = S\left(\frac{n\left(1 + \frac{v}{c}\right)}{f_s}\right) \tag{3.5}$$

If the sampling frequency f_s is adjusted by the same Doppler shift a new sampling frequency results and is given by

$$\hat{f}_s = f_s\left(1 + \frac{v}{c}\right) \tag{3.6}$$

which would produce a new n^{th} sample

$$\hat{s}_{dn} = s\left(\frac{n\left(1 + \frac{v}{c}\right)}{\hat{f}_s}\right) \tag{3.7}$$

When Eq. 3.6 is substituted into Eq. 3.7, the resampled signal is

$$\hat{s}_{dn} = s\left(\frac{n}{f_s}\right) \tag{3.8}$$

which is identical to a signal with no Doppler shift and thus signal processing can proceed as in the zero Doppler case.

Prior to signal processing, each data set was resampled at either 4 or 8 times the Doppler shifted carrier frequency for the data originally sampled at 228 Hz or 456 Hz respectively. The re-sampled values resulted from linear interpolation between actual data points as if the desired sampling frequency had been used. The average received center frequency was estimated by averaging repeated 32,768 point (140.35 second) power spectrums of the data over the one hour transmission period. The resultant time-averaged peak frequency nearest 57.0 Hz was used as the received center frequency. This value agrees with subsequent estimations of Doppler shift using the drift of the source and receiver ships from GPS data and the relative geodesic angle. Since changes in Doppler were small, no correction was made for variations in received frequency during the reception hour.

C. SIGNAL PROCESSING PROCEDURE

1. General Considerations

Data processing was performed using a proprietary signal processing applications software package named Lab Workbench which was developed by Concurrent Computer Corporation. This software provides a shell interface between a 32 channel analog-to-digital converter, the Concurrent 6400 Real-Time Unix based computer system, numerous signal processing software modules, and the optical disk data storage medium. Most importantly, Lab Workbench allows optional user-defined data processing modules which are, in effect, Fortran or C language subroutines which can operate real-time on the data as it is being collected.

Because hardware difficulties prevented the use of the real-time capabilities during the Heard Island Feasibility Experiment, all data was processed after the conclusion of the experiment, using a playback module to read the data from the storage media. The stored data was retrieved from the optical disk and processed in buffer lengths of 1 second each. The 1 second buffers of data were passed sequentially through a bandpass filter module, a demodulator module, two parallel low-pass filter modules, a sequence removal module, and finally a module which writes the processed data onto the storage media. Each of the signal processing modules as well as pre-processing steps are described in sequential order below. Figure 3.3 illustrates the time domain processing accomplished with the Lab Workbench modules. Listings of C language source code for the processor modules are included in Appendix B. Modular design of the processing

Figure 3.3 Overview of Signal Processing Steps

software allowed increased flexibility of implementation and operation in any one of the following modes:

- Setup: Specification of all run-time variables, memory allocation, and preliminary calculations. Performing routine calculations and 'housekeeping' functions in this mode speeds operations during the run mode.
- Run: Actual real-time data processing which occurs in buffer lengths of one second each until the entire one hour transmission has been processed.
- Stop: Halts the Run phase of data processing but maintains current data in the transfer buffers to allow subsequent restart.
- Menu: Allows selection of variable values such as carrier frequency, m-sequence code, phase shift, etc. during operation or execution of the Lab Workbench software.
- Drop: Same function as Stop but drops the last data transfer buffer.

2. Bandpass Filter Module

The power spectrum of the phase encoded m-sequence is the convolution of the carrier power spectrum and the m-sequence code power spectrum as developed in Section 2.C.3. Therefore the retrieved data, which was hardware bandpass filtered 30 to 80 Hz prior to recording was again bandpass filtered. This second bandpass filter passed the band from 45 to 69 Hz using a software implementation of a classical Hamming windowed, 20^{th} order, linear-phase, finite impulse response (FIR) filter. Received data was passed in a 24 Hz band corresponding to the power spectrum of the biphase modulated 57 Hz carrier. The filter structure is the general tapped delay-line filter described by the difference equation

$$y_n = b_0 x_n + b_1 x_{n-1} + \ldots + b_{20} x_{n-20} \quad (3.9)$$

for data vector **x** and weighting coefficients $\mathbf{b}_n$ obtained using the fir1 function in Matlab. Matlab is a proprietary software product developed by The MathWorks, Inc., South Natick, MA. Although the C-language Lab Workbench implementation of this filter is listed in Appendix B as bandpass.c, any other filter type can logically be substituted if it maintains linearity in phase.

3. Demodulation

To retrieve the digital code from the biphase modulated carrier signal, the received data must be demodulated. The object of complex or quadrature demodulation is to isolate the real bandpass signal on the positive frequency axis and translate it to a complex lowpass signal. If $\mathbf{f}_d$ is the downshift frequency in Hz and **t** is time in seconds, the demodulation can be accomplished by multiplying the input data by a complex rotator given by (Birdsall and Metzger, 1990)

$$d_t = s_t e^{-i 2 \pi f_d t} \quad (3.10)$$

where $\mathbf{s}_t$ represents the input signal and $\mathbf{d}_t$ is the demodulated output signal. When the sampling rate $\mathbf{f}_s$ is precisely the fourth harmonic of the carrier frequency $\mathbf{f}_c$, as it is for the re-sampled data, $\mathbf{f}_d\mathbf{t}$ changes by exactly one-quarter cycle with each successive sample and the values of the downshifting complex rotator $e^{-i2\pi(fc/fs)n}$ being 1,-i,-1,i;1, -i, -1, i; etc. The complex multiplication then results in successive polarity changing and alternating assignment to real and imaginary parts. The result is a real and imaginary train of pulses corresponding to the m-sequence code at a baseband of 0 to 11.4 Hz. Since the demodulator is not synchronized with the transmitter, it must demodulate the signal

without the benefit of phase information. Therefore, to recover all of the magnitude of the signal in the baseband, the signal must be multiplied by both the cosine and sine at the carrier frequency. The digital implementation of the real part of the demodulation is given by

$$r_n = s_n \cos\left(\frac{2\pi f_c n}{f_s}\right) \tag{3.11}$$

where $\mathbf{r_n}$ is the n^{th} demodulated real data point. A listing of the C-language code is given in Appendix B as demod.c. The digital formulation of the imaginary part, $\mathbf{i_n}$, is similarly

$$i_n = s_n \sin\left(\frac{2\pi f_c n}{f_s}\right) \tag{3.12}$$

Ideally, the downshifting frequency $\mathbf{f_d}$ would be the received signal carrier frequency which would translate the signal spectrum perfectly to it's baseband. However, since the signal was re-sampled at 4 or 8 times the measured $\mathbf{f_c}$, we can use any combination of $\mathbf{f_c}$ and $\mathbf{f_s}$ in equations 3.11 and 3.12 such that $\mathbf{f_s/f_c=4}$ **or** $\mathbf{8}$ and still maintain correct synchronization for the complex demodulation rotator. Since the transmitter design carrier frequency was 57.0 Hz, the ideal sampling frequency of either 228 Hz or 456 Hz were used for $\mathbf{f_c}$ and $\mathbf{f_s}$ respectively.

4. Low Pass Filter

Both the in-phase and quadrature components from the demodulator were low pass filtered passing the band from 0 to 12 Hz. The filters were a software implementation of a classical Hamming windowed, 20^{th} order, linear-phase, finite impulse response (FIR) filter. The passed band from 0 to 12 Hz. corresponded to the power spectrum of the baseband of the biphase modulated 57 Hz carrier. The filter structure is

again the general tapped delay-line filter described by the difference equation (Eq. 3.9) with vector **x** and coefficients b_n obtained using the fir1 function in Matlab. Although the software implementation of this filter is listed in Appendix B as lowpass.c, any other filter type can logically be substituted if it maintains linearity in phase.

5. Sequence Removal

Correlation of the received data with the transmitted maximal length sequence code is performed by a software adaptation of the Fast Hadamard Transform. The permutation matrices discussed in Chapter II are calculated for the specific m-sequence code and stored during the setup mode of the sequence removal module.

During run-time execution of the module, dual inputs of real and imaginary data are input and multiplexed into a large storage array. Since the Hadamard processing must be performed on an entire sequence, the storage array will hold 22.4, 44.8, 89.8, or 179.6 seconds of data prior to processing, depending on the specific m-sequence code. Remember that the transmitted signal had 5 cycles per digit and the signal was sampled at $\mathbf{4f_c}$ or $\mathbf{8f_c}$ so there were 20 data points collected per digit for the 228 Hz sampled data and 40 points per digit for the 456 Hz sampled data. When the signal was demodulated the number of data points per digit grew by a factor of two, since the demodulated signal has real and imaginary parts. The incoming real **r** and imaginary **i** data buffers are interleaved into the storage array by alternating $\mathbf{r}_0$, $\mathbf{i}_0$, $\mathbf{r}_1$, $\mathbf{i}_1$,.....$\mathbf{r}_{39n}$, $\mathbf{i}_{39n}$ for an n-digit sequence sampled at 228 Hz.

Since the Fast Hadamard Transform only requires one data point per digit, by the time a single m-sequence has been received the storage array actually contains 40 or

80 interleaved m-sequences depending on the sampling rate. Each sequence is sequentially permuted according to the scrambling array, processed by the FHT, and then permuted again according to the unscrambling array. When the Fast Hadamard Transform is performed successively on each interleaved sequence in the storage array, the result from an ideal signal is 40 or 80 adjacent peaks at the 0-shift or correlated digit. This oversampling causes a flat-topped, one digit wide correlation peak vice the 'pointy' autocorrelation peak developed in Chapter II.

As shown in Figure 2.7, the autocorrelation function for an m-sequence has a negative DC level for all digit positions except the first. Therefore, the zeroth position of each sequence contains the average DC level and can be calculated and removed from all other positions. The complex level adjustment C_{cor} is related to the average DC level L_{avg} by

$$C_{cor} = f_1 L_{avg} \tag{3.13}$$

and f_1 is given by the relation

$$f_1 = j \frac{(N+1)\tan(\theta) + j(N-\tan^2(\theta))}{N^2 + \tan^2(\theta)} \tag{3.14}$$

where N is the m-sequence digit length and Θ is the phase modulation angle. All the parameters of equation 3.15 are known and so f_1 is calculated only once during the setup phase of the sequence removal module. Bias removal then requires one complex multiplication per sequence during the run mode to calculate C_{cor} and the correction can be subtracted from each value in the sequence.

Since the processed data array is interleaved with real and imaginary parts of the transformed signal, it is a trivial matter to compute the magnitude $\mathbf{mag_n}$ in the time domain by

$$mag_n = \sqrt{r_n^2 + i_n^2} \tag{3.15}$$

The ideal processed signal now is a time series which has zero values at all points except the correlated digit. At the correlated digit, there will be a sequence of pulses whose combined width is one digit wide (0.087 seconds) and whose amplitudes are equal to the number of digits in the sequence. For real data, the model output is similar with the addition of randomly distributed white noise present in all digits as shown in Figure 3.2.

Although a Doppler shifted input signal can be correctly demodulated without resampling, the bandpass digit sequence would still be either expanded or compressed in time. The Fast Hadamard Transform scheme used in this thesis assumes exactly 20 digits per sample (40 for the 456 Hz sampled data) and only one sample per m-sequence digit is transformed at a time. Thus any expansion or contraction of the time series greater than one sample interval would cause a loss of a digit. A single output of the FHT process would have a reduction in amplitude of **1/N** for each shift of one sample length where **N** is the number of digits in the sequence. Because the interleaved data produces 20 (or 40) adjacent output peaks, the reduction occurs first on the end peaks and then successively toward the opposite end as the Doppler shift increases. The result is a narrowing of the pulse at the top with increasing Doppler and an eventual decrease in the pulse amplitude for Doppler time shifts exceeding one digit width T_c of 0.0877 seconds over the length of the sequence. The Doppler frequency shifts required to cause the loss

of one complete digit range from 0.028 Hz (1.43 kts) for the 2047 sequence to 0.2224 Hz (11.5 kts) for the 255 sequence. Therefore, it is imperative that the data is resampled to remove Doppler time alteration and preserve a constant number of samples per digit.

D. POST-PROCESSING ENHANCEMENTS

1. Time Averaging

The primary objective of time domain processing is to obtain a correlated pulse whose amplitude is sufficiently greater than the average noise level to allow confident detection. Since the m-sequence is transmitted repeatedly during the one-hour transmission period, the repetitive sequences can be averaged incoherently by simply summing the corresponding data points of successive sequences. As the number of averages increases, a stable correlated signal pulse will remain at the same relative position in the time series but the Guassian noise will tend toward an average value of zero. The sequence removal module facilitates averaging by retaining leftover buffer data when a full sequence has been received so that it can be used at the beginning of the following sequence. In this way a continuous stream of data is processed as it was received.

2. Visual Integration

Because the received signal is often very weak even after significant processing to maximize gain, it is useful to display the resultant data in a manner conducive to easy interpretation and recognition of the signal. It is particularly helpful to display the graphical output of several time series such that corresponding times are

aligned. In this way, any signal which may be marginally detectable in a single time series may stand out as a trend among several sequential series.

The Fortran program hafton.f which is listed in Appendix B was written to take advantage of this visual integration between sequential time series. It produces a two-dimensional plot with m-sequence time series on the horizontal axis and successive m-sequences 'stacked' above in sequential order. The amplitudes are represented by halftones of various gray shades with higher amplitudes indicated by darker shading. These plots can be further enhanced by plotting a function of the data such as log, sine, cosine, etc. to accentuate the extreme or mean regions of data amplitudes.

An additional method of achieving visual integration is the use of a 'waterfall' display where successive time or frequency domain plots are 'stacked' and recurring peaks become apparent. The matlab program mesh.m was used to generate the waterfall type displays included in this thesis.

IV. RESULTS

A. DATA COLLECTION

The first objective of this thesis was to successfully deploy a vertical array in the optimum reception zone and record the sampled acoustic environment at a rate higher than the Nyquist frequency. This goal was met cooperatively between the Naval Postgraduate School, the Monterey Bay Aquarium Institute, Woods Hole Oceanographic Institution, and Massachusetts Institute of Technology. The output of the transducers was sampled at either 228 or 456 Hz, bandpass filtered, and optimally amplified to maximally utilize the dynamic range of the analog-to-digital converter as discussed in Chapter III.

At 01261525 (1525 Greenwich Mean Time on 26 January 1991) *R/V Point Sur* was on station near the northern edge of the predicted reception zone, recording all transmissions until the array cable became entangled in the ships screw at 01300205 GMT. During this period the *R/V Point Sur* recorded 12 transmissions and steamed between reception periods from 35° 41'N, 123° 01'W, to 35° 30'N, 123°05'W in order to sample a cross section of the predicted reception zone. Following a swift transit to Monterey in order to free the fouled screw and an admirable cable splicing repair effort led by K. Von der Heydt of WHOI and K. Lashkari or MBARI, *R/V Point Sur* returned to a point further into the predicted reception zone at 34° 15'N, 122° 00'W by 01312108 GMT. Unfortunately, by this time the transmitting vessel, RV Cory Chouest, had become encumbered by heavy seas and the reliability of it's overdriven transducers was

deteriorating rapidly. Two abbreviated receptions were recorded between 01312108 GMT and 02010039 GMT at the new location before untenable conditions near Heard Island forced conclusion of the Experiment. The 32 channels of digitized data were sampled and recorded to 1 Gigabyte Tahiti ZCAV optical disks at 58.386 kilobytes per second for the 228 Hz sample rate and 116.736 kilobytes per second for the 456 Hz sample rate.

Daily sound speed measurements were recorded using an XBT and representative sound velocity profiles are shown in Figure 4.1. In addition array depth and tilt data was

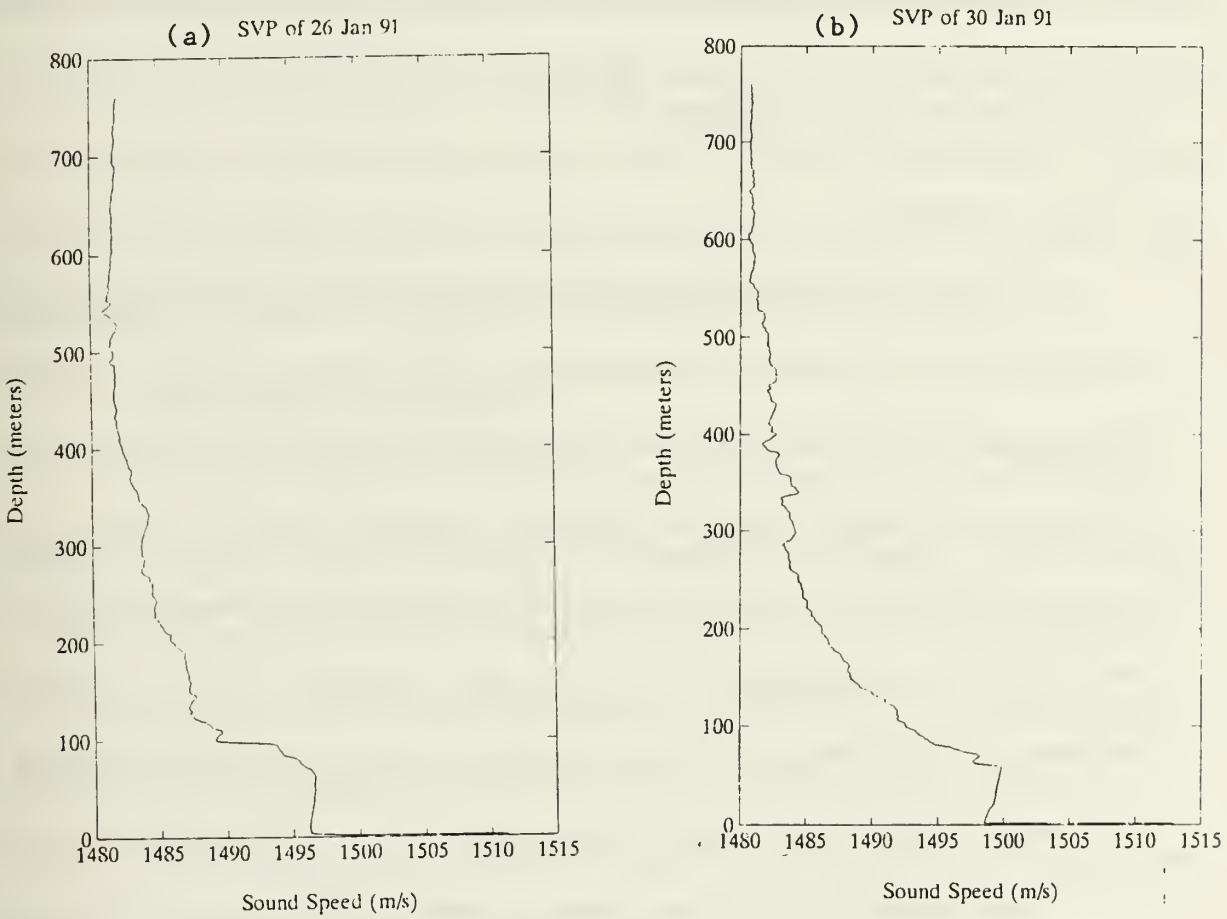

Figure 4.1: Sound Velocity Profiles.

recorded from upper and lower instruments on the array. This data is used to determine hydrophone location relative to the sound channel axis located at the sound velocity minima. The top hydrophone was nominally at 345 meters depth. Follow-on data analysis will use the array depth and tilt data to implement modal beamforming with all 32 hydrophones. Beamforming can theoretically provide 10 log(32) or 15 dB of array gain and a realistic expectation of between 6 and 10 dB. In addition, if the hydrophone outputs are properly weighted, the array will act as a mode filter to discriminate between various normal modes of propagation and more accurately depict the propagation method.

B. SIGNAL PROCESSING

1. Objective

The second goal of this thesis was to develop and implement signal processing software capable of optimizing the characteristics of the transmitted signal. The software was designed to operate in real time on an incoming data stream using a Concurrent 6400 data acquisition computer with Lab Workbench applications software. To this end, the programs in Appendix B which were discussed in Chapter III have been developed. All processing was implemented with user function modules in the Lab Workbench environment. All user function modules were written in the C programming language and can be utilized real-time as the data is being collected. The 32 channel input data stream can be demultiplexed with a channel selector switch to permit analysis of individual channels or, if desired, arithmetically combined to sum individual channels. Piping data between modules is facilitated by connecting the input and output boxes of appropriate

modules on-screen with a click of the left mouse button. In addition, user interface with the running program was enhanced with on-screen menus to permit selection of key parameters for each function. The on-screen menus are activated by clicking the left mouse button while the cursor is positioned on the module of interest.

2. Test Data

The signal processing programs were tested with an artificial data set which was recorded onto audio cassette tape and then played back through a single channel of the data collection system. Test data consisted of 66 consecutive sequences of 255 digit m-sequence code (octal law 537). The inherent noise of the audio tape and variability of the tape drive speed provided both a source of near-white noise and frequency wavering similar to the Doppler shift of the drifting source and receiver. The results of the performance, and sensitivity testing using the test data are demonstrated below.

3. Resampling and Clipping

Figure 4.2 compares the signal band power spectrum of the 01270005 m-sequence raw data to the same data after resampling and clipping. The C function resamp.c clips and resamples 32 channels of multiplexed data at a rate exactly four times the Doppler shifted carrier frequency as discussed in Chapter III. The function clips the data at a magnitude which is a user defined factor times the standard deviation (σ) of a local 10 second time series. It allows the user to define both the Doppler shifted resampling frequency and the clipping factor from an on-screen menu. All

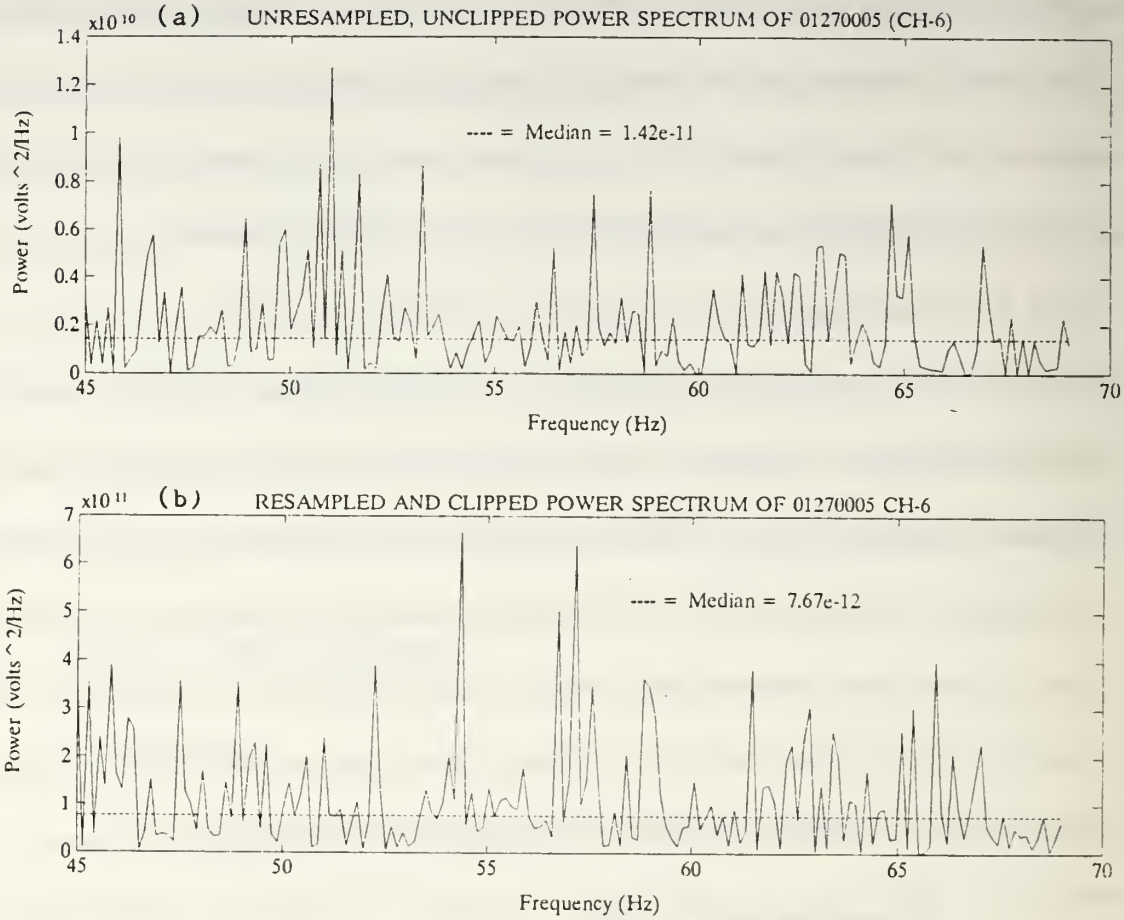
x10 10 (a) UNRESAMPLED, UNCLIPPED POWER SPECTRUM OF 01270005 (CH-6)
---- = Median = 1.42e-11
Power (volts ^ 2/Hz)
Frequency (Hz)
x10 11 (b) RESAMPLED AND CLIPPED POWER SPECTRUM OF 01270005 CH-6
---- = Median = 7.67e-12
Power (volts ^ 2/Hz)
Frequency (Hz)

Figure 4.2: Results of Clipping and Resampling 01270005 Raw Data.

data analyzed for this thesis was clipped at the default value of 2σ and resampled at exactly 4 times the median carrier frequency as determined by spectral analysis. A 32,768 point FFT was used in all spectral analysis in this thesis. Resampling simplifies implementation of the demodulation procedure by minimizing the effects of relative motion induced Doppler frequency shifts of the received carrier frequency. The clipping of this particular time series resulted in approximately a 3 dB decrease in the median

4. Bandpass Filtering

Figure 4.3 displays the filter response of the 20th order FIR bandpass filter for both 228 and 456 Hz sample rates superimposed on the normalized magnitude envelope

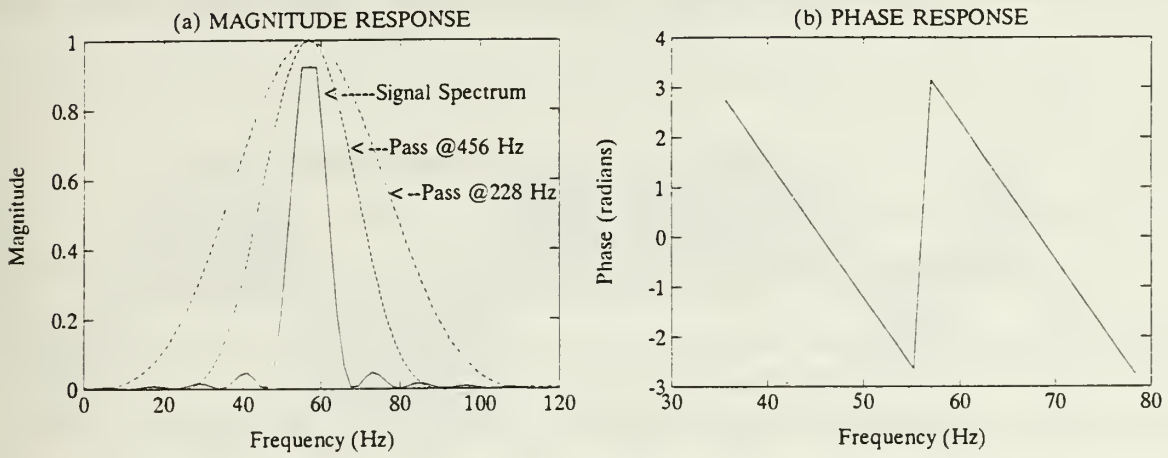

Figure 4.3: Magnitude and Phase Response of 20th Order Bandpass Filter.

of the m-sequence signal. The filter passes frequency components between 45 and 79 Hz. It is readily apparent that removal of unwanted out of band noise will significantly improve the signal-to-noise ratio of the processed data since unwanted noise in the time domain is comprised of a wide spectrum of frequencies. Bandpass filtering was performed with the function bandpass.c. The program allows user selection of the sample frequency (which determines the filter coefficients) from an on-screen menu.

5. Demodulation

The complex demodulation of the received signal into its quadrature and cophase components at baseband frequency was also implemented with a Lab Workbench user function module. It is listed in Appendix B as fdemod1.c. Both the real and imaginary parts of the signal are produced as output streams for further manipulation,

thereby preserving all of the signal information for maximum processing gain. The user can select carrier frequency and sample frequency from an on-screen menu. Figure 4.4 shows a 1 second portion of the test data time series after processing with the demodulator. This one second series contains 11.4 digits each of length 0.0877 seconds.

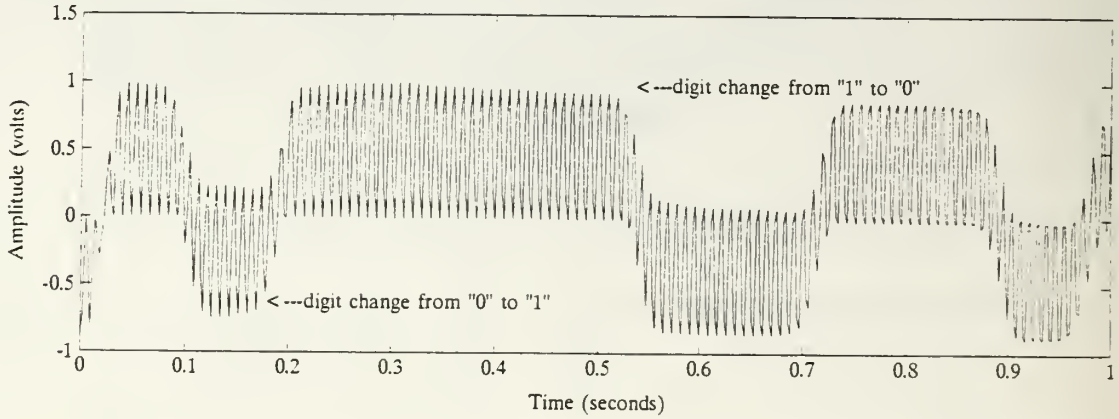

Figure 4.4: Real Part of Demodulated 255 M-Sequence Test Data.

6. Lowpass Filtering

The output of the demodulator in Figure 4.4 exhibits some high frequency (57 Hz) ripple components in the time series. Since the m-sequence has been translated to its baseband frequency (0 to 11.4 Hz), these higher frequency components can be effectively removed without impairing the signal level by lowpass filtering. As with the bandpass filter, removal of out of band noise will increase the SNR of the processed time domain signal. The filter was designed to pass frequency less than 12 Hz, including the 11.4 Hz. envelope of the demodulated signal. The program allows selection of the sample rate from an on-screen menu. Figure 4.5 displays the magnitude and phase response of the lowpass FIR filter superimposed on the baseband envelope of the m-sequence.

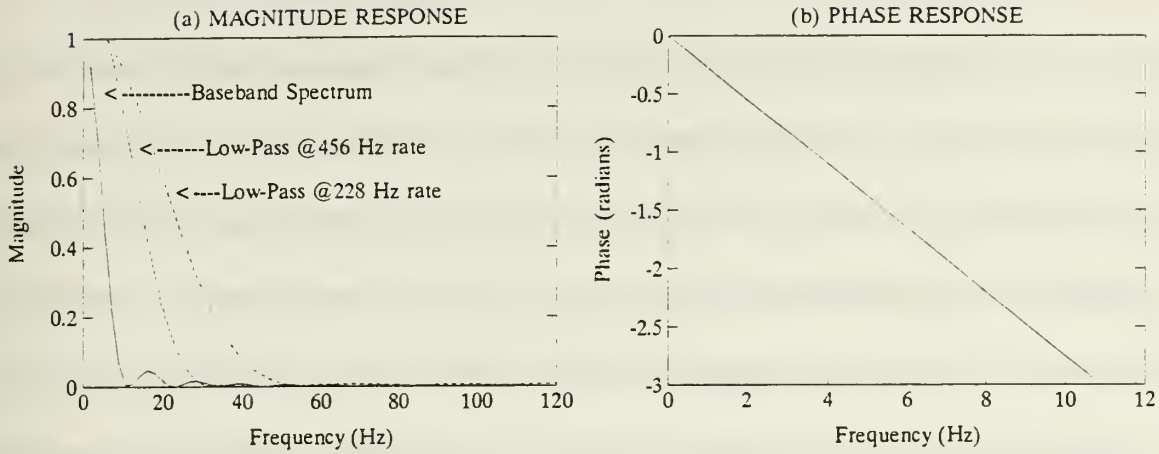

Figure 4.5: Magnitude and Phase Response of 20th Order Lowpass Filter.

Figure 4.6 shows the same 1 second portion of demodulator output as Figure 4.4 but after processing through the lowpass filter. Note that this figure represents only one second of a 23.368 second sequence which is comprised of a sequence of 255 ones and zeroes. Additionally, note that Figures 4.4 and 4.6 show only the real part of the signal. Both the real and imaginary demodulated streams are output for simultaneous subsequent processing.

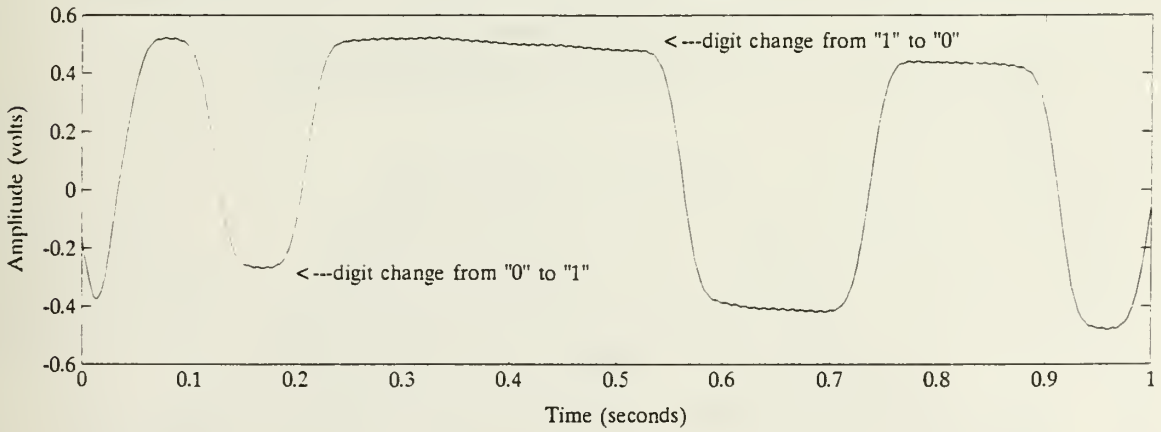

Figure 4.6: Lowpass Filtered, Demodulated 255 Test Data.

7. Sequence Removal

Both lowpass filtered outputs of the demodulator are piped into the sequence removal function, fmet.c. The data is passed in one second buffers into a storage array until an entire m-sequence has been accumulated for processing. The program then performs the Fast Hadamard Transform on the 40 or 80 redundant and interleaved m-sequences (both real and imaginary) as discussed in Chapter III. The real and imaginary components are then combined to produce the magnitude squared (intensity) of the processed signal. Figure 4.7 shows the sequence remover output for sequence number 58 of the 255 m-sequence test data. A closeup view of the same sequence illustrates the

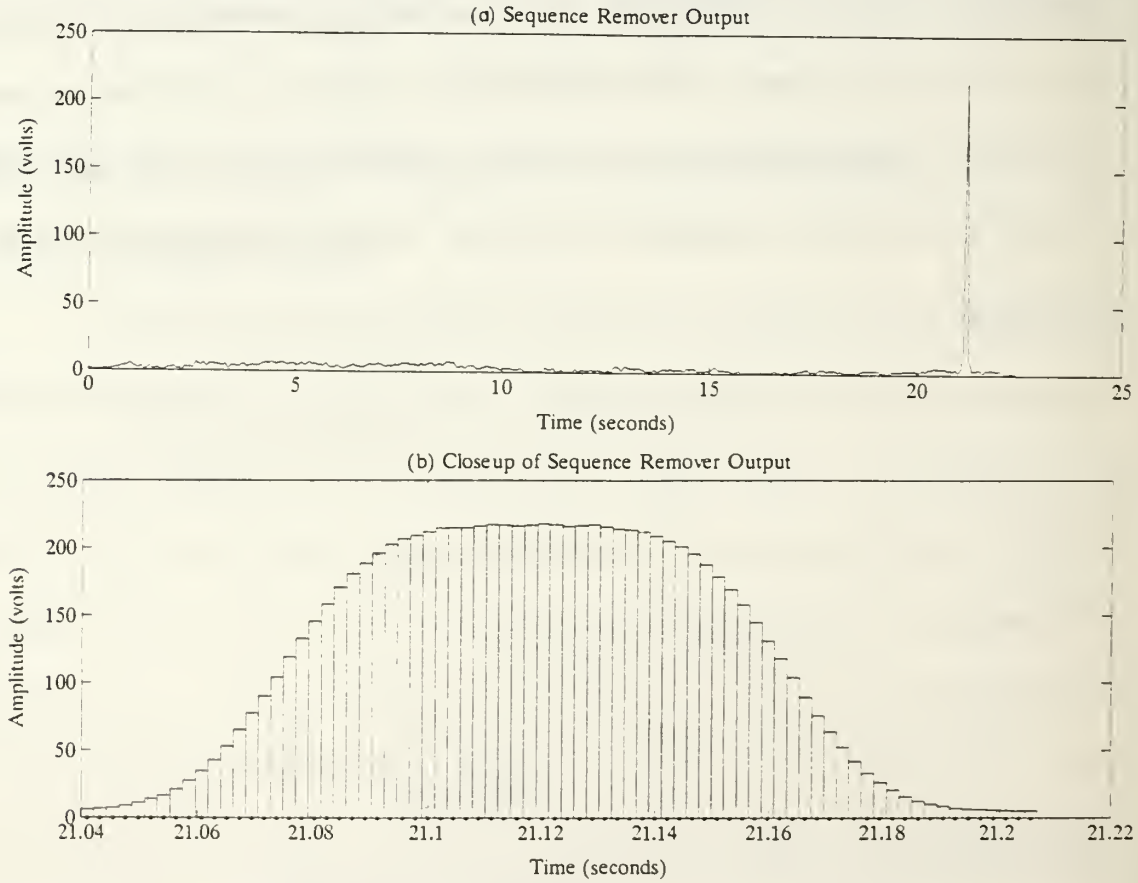

Figure 4.7: Sequence Remover Output for Sequence #58 of 255 M-Sequence Test Data.

contribution of the 40 sample points during the 0.087 second correlated digit. The effectiveness of the sequence remover is dependent on the correct demodulation frequency as discussed in Chapter III. The variation of the audio cassette tape speed as the test data was digitized created a Doppler-like variation of signal frequency. Figure 4.8(a) is a plot of the peak sequence remover output values for 66 consecutive sequences showing the variation in amplitude with time. Figure 4.8(b) shows the deviation from demodulation frequency (57.003 Hz) which can be visually correlated with the output magnitude in Figure 4.8(a). As the actual carrier frequency deviates from the demodulation frequency,

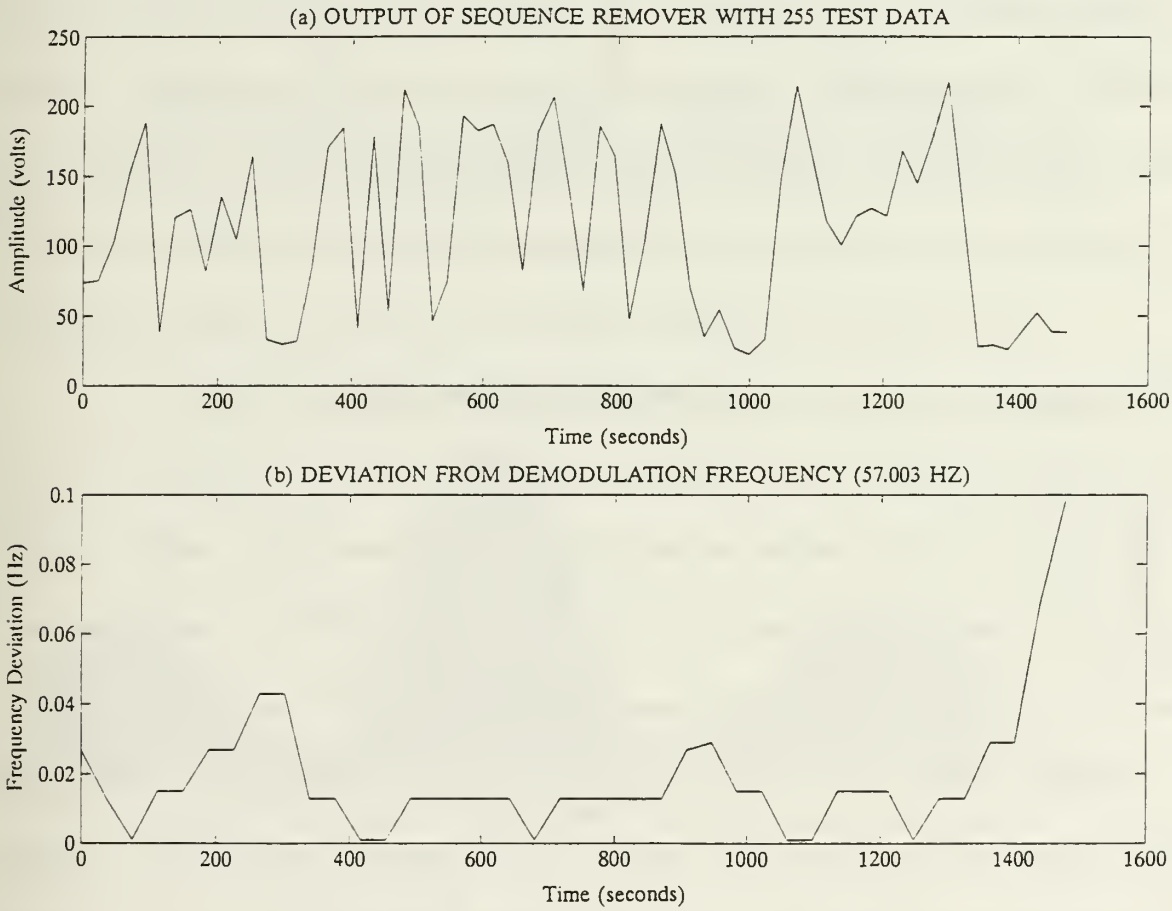

Figure 4.8: Frequency Deviation and Sequence Remover Output vs Time.

the energy which would ideally be correlated into a single digit is spread across the time series as noise, which is illustrated in Figures 4.9(a and b). Note that time 1297 seconds (sequence #58 in Figure 4.7) had a deviation of about 0.015 Hz (equivalent to .395 m/s relative motion) while time 962 seconds (sequence #43 in Figure 4.9) had a deviation of over 0.03 Hz (0.789 m/s relative motion).

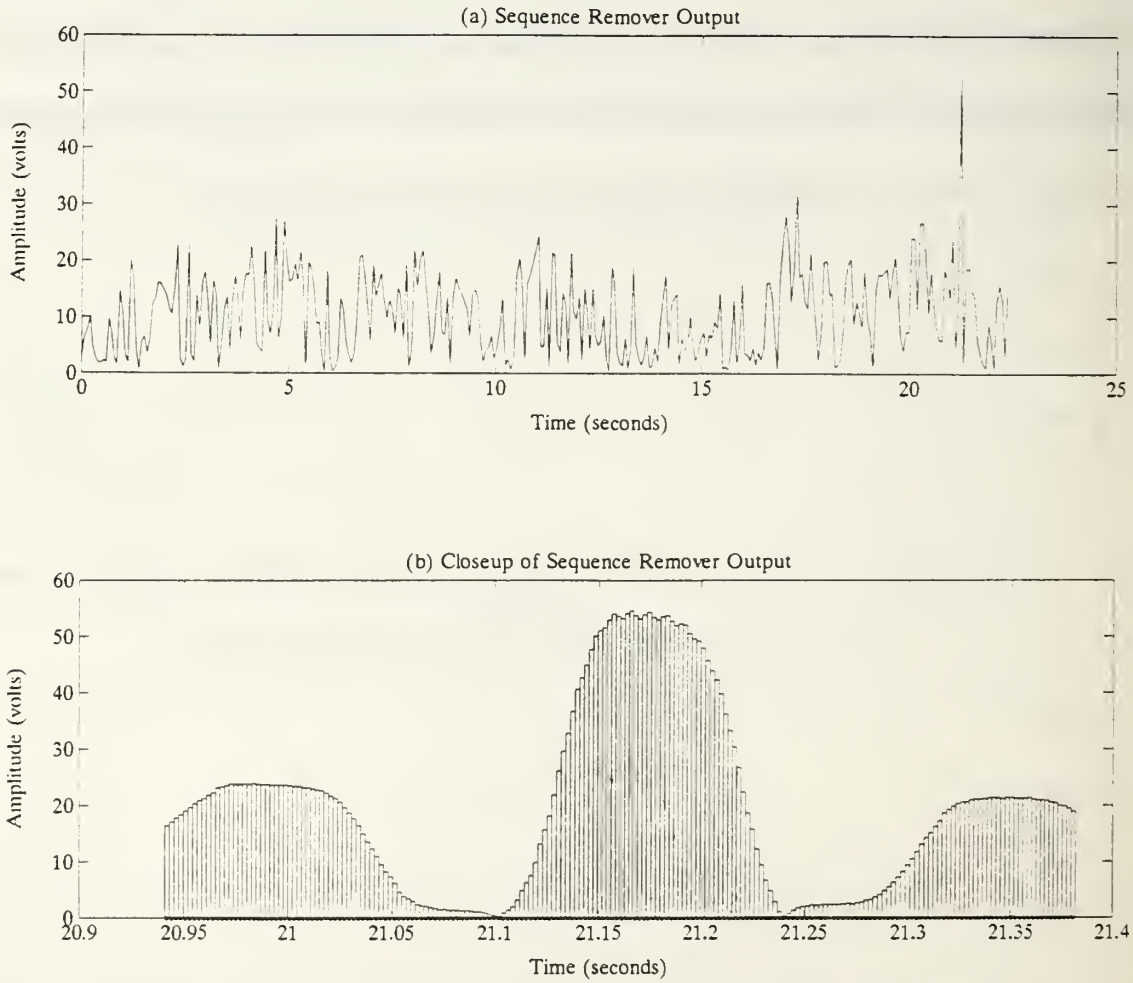

Figure 4.9: Sequence Remover Output for 255 Test Data (Sequence #43).

8. Averaging and Display

The C function spewrit.c performs several analytical functions on the data. It scans the 30 frequency bins centered on 57.0 Hz to select the bin with the highest amplitude. This corresponds to a frequency scan from 56.903 to 57.097 Hz which is the Doppler frequency shift resulting from array velocities along the propagation path equal to 0 +/- 5.06 knots. The program writes an output file containing the peak amplitude of the scanned bins, the corresponding frequency of the peak bin, and the average noise level in a 24.8 Hz band centered at 57.0 Hz. Note that if the carrier signal level is lower than a noise bin in this band an incorrect value will be selected which will produce errors in both the carrier frequency and carrier amplitude estimation. Therefore, median values are used in our estimates because the median is less sensitive to outliers than the mean. These output files are in the form of a channel-time matrix and can be easily used to estimate the signal-to-noise ratio and frequency stability of the carrier wave.

Figure 4.10 is a gray-scale plot of the same 66 255-digit m-sequences of the test data, vertically stacked so that corresponding digits of separate sequences are aligned. When a signal level is much smaller than the ambient noise level, as it was for the Heard Island Feasibility Experiment, it is often advantageous to coherently average several sequences or FFT's so that random uncorrelated noise will tend to cancel and correlated signal energy will stand out. The C function, writer.c, writes the repetitive m-sequence outputs of the sequence removal module or the repetitive autospectral density outputs of the power spectrum module to a file such that corresponding times or frequencies are aligned. The values in the output file of writer.c can then be easily

plotted as a three dimensional surface, averaged, and plotted as a two dimensional subset of the file.

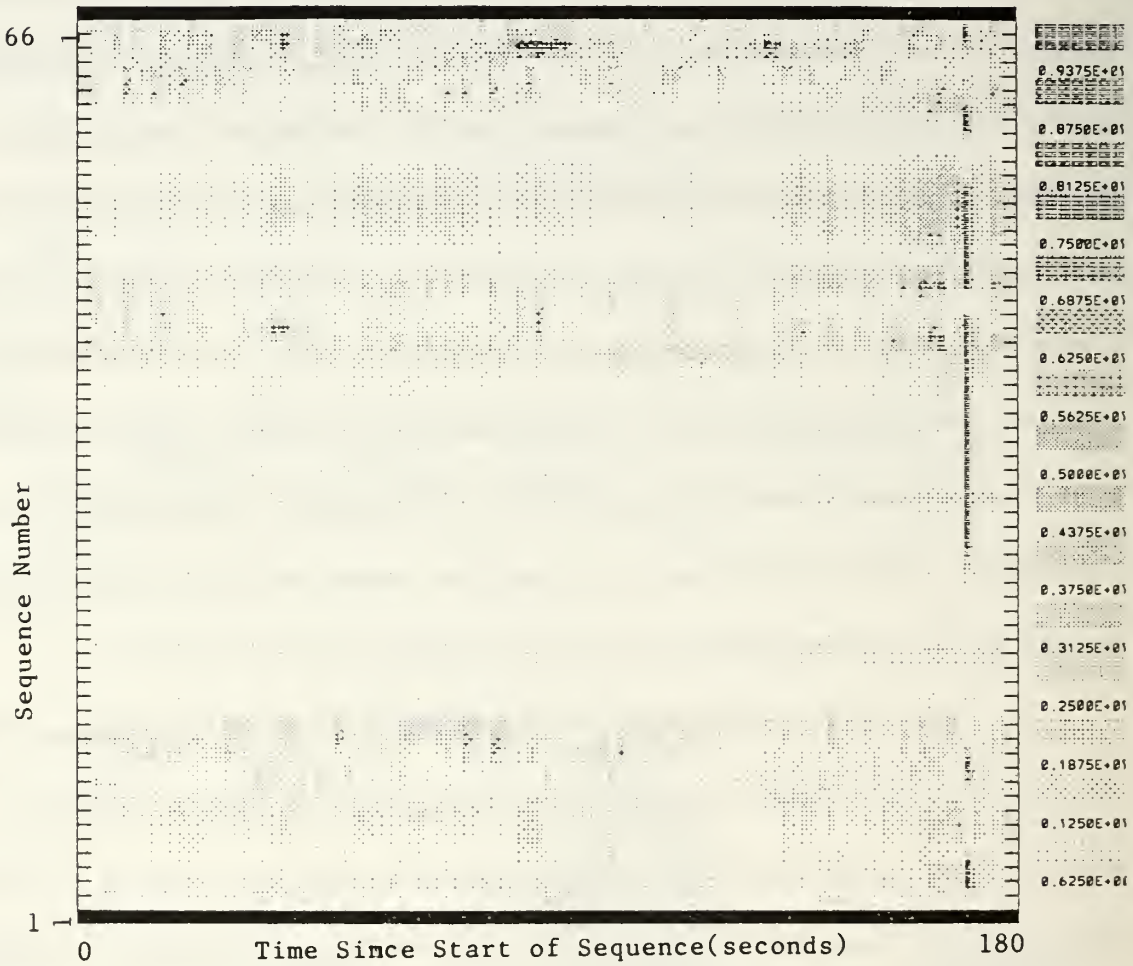

Figure 4.10: Gray-Scale Plot of Sequence Remover Output for 66 255-digit M-Sequences of Test Data.

C. PRELIMINARY DATA ANALYSIS

1. Objective

The goals of the preliminary data analysis was to determine the signal-to-noise ratio of the received signal and determine whether it was sufficient (greater than 18 dB post-processing) to permit arrival time estimation within the prescribed 10 millisecond accuracy for use in a long term global warming experiment. Additionally, sequence removal was conducted in an effort to directly analyze arrival structure and stability. In particular, we desired to find the maximum coherent integration time of the reception which, in turn, would determine the maximum processing gain.

2. Frequency Domain Processing

a. Autospectral Density

Autospectral density (or power spectrum) estimation was performed using the modified parallelogram method [Reference 28]. The process yields the mean square intensity or power of the signal as a function of frequency and is available as a packaged module of Lab Workbench. All frequency domain processing utilized the maximum Fast Fourier Transform input points (**N=32768**), Hamming windowing, and 50% overlap of adjacent FFT's. The sample interval (Δt) was 4.386 and 2.193 milli-seconds for the 228 Hz and 456 Hz sample rates respectively and the corresponding FFT block lengths, given by $(N-1)\ \Delta t$, were 143.715 seconds and 71.857 seconds respectively. The frequency resolution or binwidth of the power spectrum, (B_e) given by $1/\Delta$, is then 6.958 and 13.916 millihertz, again dependent on the sample rate.

Analysis in the frequency domain provided a quick assessment of equipment operation as well as an indication of relative signal and noise levels. Performance of individual channels can be estimated from the deviation of their output from the median levels. Channels 14-19, 21-23, 28, 29, and 31 had indications of extremely high or low output. Only channels whose output was consistently within 3 standard deviations of the array median were used to estimate signal and noise values. Figure 4.11 shows the median peak spectral value (of all valid channels) for the 01270222

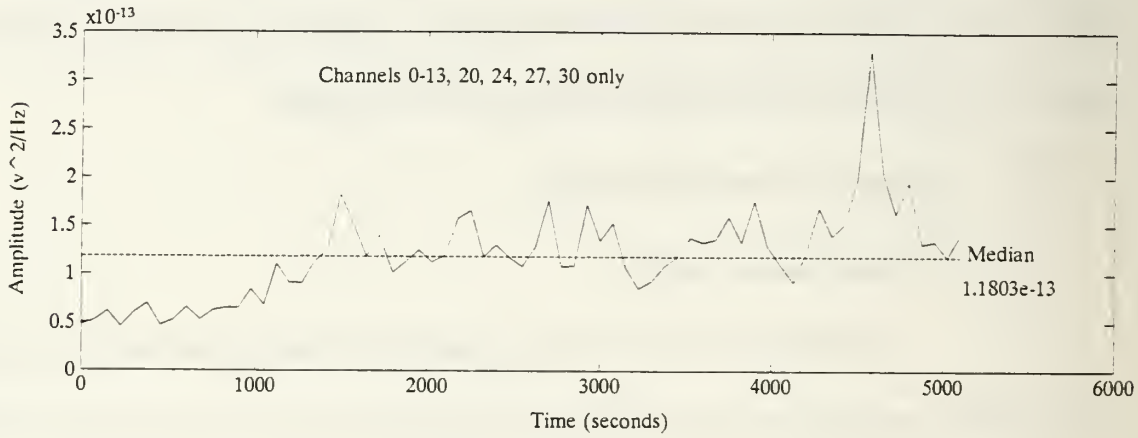

Figure 4.11: Median 57.0 Hz Spectral Peak of 01270222 CW Signal.

CW signal in a 0.193 Hz band centered at 57.0 Hz (corresponding to +/- 5 kts Doppler) as a function of time. The median channel spectral noise value in a band from 45 to 69 Hz is plotted in Figure 4.12. Although band noise is not strictly applicable for a CW signal, using a 24 Hz band for noise summing provides a reasonable estimation of expected noise levels for the m-sequences. The noise value was obtained for each power spectrum by summing the values in each frequency bin of the 24 Hz band and then dividing by the number of bins (862). Finally, the median value of all valid channels was

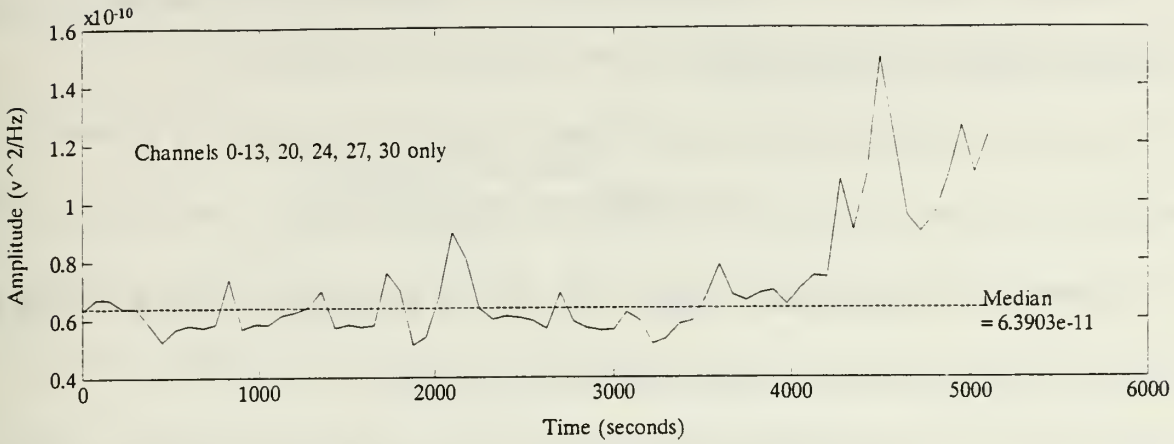

Figure 4.12: Median 12 Hz Band Noise of 01270222 CW vs Time.

plotted as a function of time. Taking the median again (of the median channel values) provides an overall median value of 1.18×10^{-13} volts^2/Hz for the signal level and 6.39×10^{-11} volts^2/Hz for the noise value. These values can be converted to acoustic power levels of 19 dB rel 1μPa for the signal and 46.4 dB rel 1μPa for the noise, as discussed below. Transmission loss over the mean acoustic path would then be about 213 dB - 19 dB = 194 dB.

b. Signal-to-Noise Ratio

The rms power in a given frequency bin is obtained by multiplying the autospectral density G_{xx} times the frequency binwidth, $(\mathbf{B_e})$. Since transducer output voltages were recorded, the measured rms power is in units of volts squared per Hertz and can be related to acoustic power by the sensitivity of the hydrophones (-170 Db rel 1μPa or 3.162×10^8 μPa/V). Acoustic power is the pressure force times the velocity of the pressure wave (which is in phase with the force at large distances from the source).

Therefore output of the autospectral density estimator can be converted directly to acoustic power (**P**) if desired by

$$P = 10\log\left[\frac{(3.16x10^8)^2 B_e G_{xx(f)}}{1\ \mu Pa^2}\right] \qquad (4.1)$$

which yields **P = 148.4 dB + 10log G_{xx}** for the 228 Hz sample rate and **P = 151.4 dB + 10log G_{xx}** for the 456 Hz sample rate in dB rel 1μPa^2.

Determining the SNR of the CW signals is simply a matter of measuring the signal power in the correct Doppler shifted carrier frequency bin and then dividing the result by the total noise power in the FFT band (summing the noise power in all bins). This is accomplished by simply dividing the channel median of the spectral peak matrix (Figure 4.11) by the channel median of the band noise matrix (Figure 4.12). Figure 4.13 illustrates the result. The overall median signal-to-noise ratio for the reception can then be calculated as 10 log (median signal / median noise) which equals -27.6 dB. Recall that the total noise is significantly reduced by clipping and filtering, increasing the SNR.

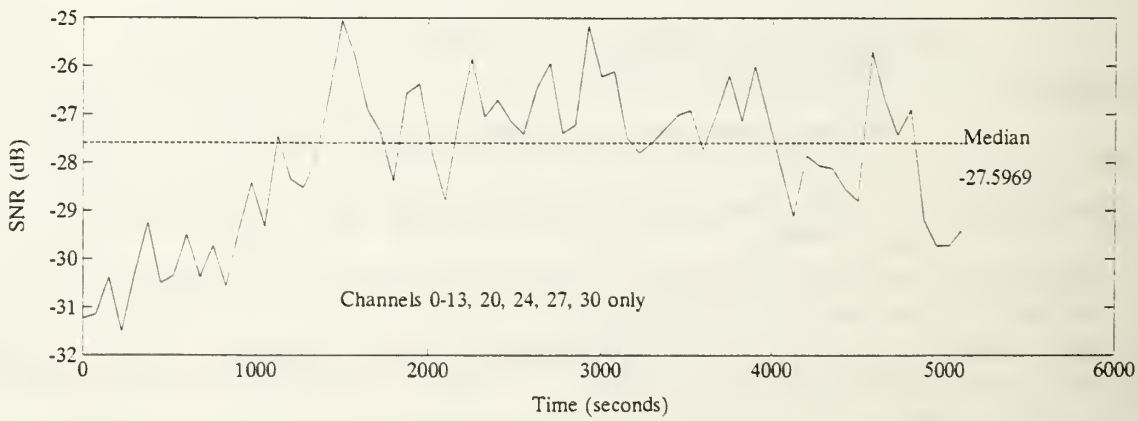

Figure 4.13: Median SNR of 01270222 CW Signal.

c. Arrival Structure

Continuous wave signals contain 100% of the power in the carrier and are therefore the optimum waveform for resolving the spatial distribution of energy over the water column. To analyze the arrival structure of the signal the matrix of carrier bin autospectral density values was formed. The matrix dimensions are 32 channels by N FFT periods, where N is the number of FFT periods in the reception, so that each matrix element contains the autospectral density for the corresponding channel and FFT period. This single arrival structure matrix contains both the temporal and spatial variability information for a single reception. The median value of the 18 valid channels, plotted in Figures 4.11 through 4.13, provides the temporal arrival pattern of the power level during the entire reception. A median value of the N FFT's produces a plot of spatial variability in signal power level over the depth of the array. The time averaged FFT is shown in Figure 4.14. When compared to the sound velocity profile, this spatial

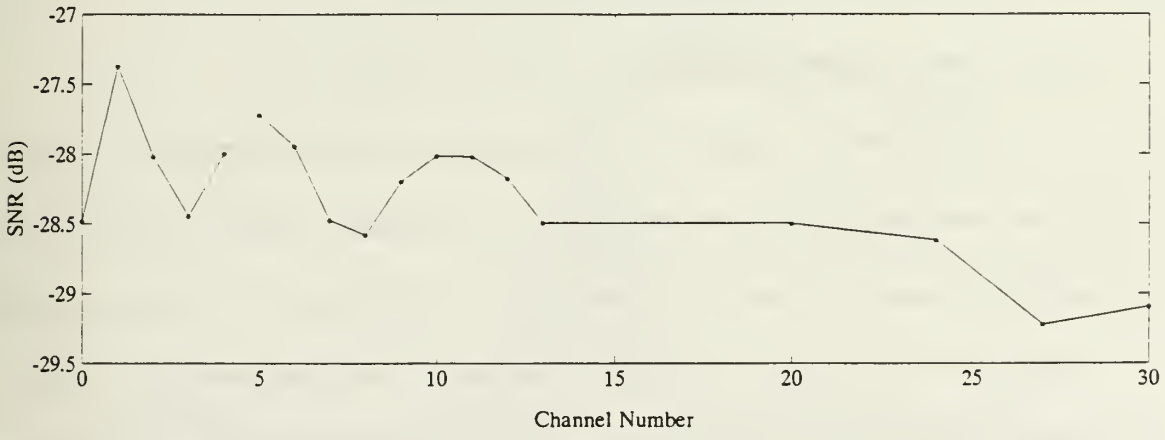

Figure 4.14: Median SNR (in 24 Hz band) vs Channel for 01270222 CW

distribution of energy might provide a clue to the predominant modal composition of the signal and thus the primary propagation means.

The median channel frequency of the 01270222 CW signal is indicated in Figure 4.15. This figure demonstrates that the frequency was remarkably constant considering that both arrays were subject to motion. Again, the large deviations from the median are probably attributable to inaccurate spectral peak selection.

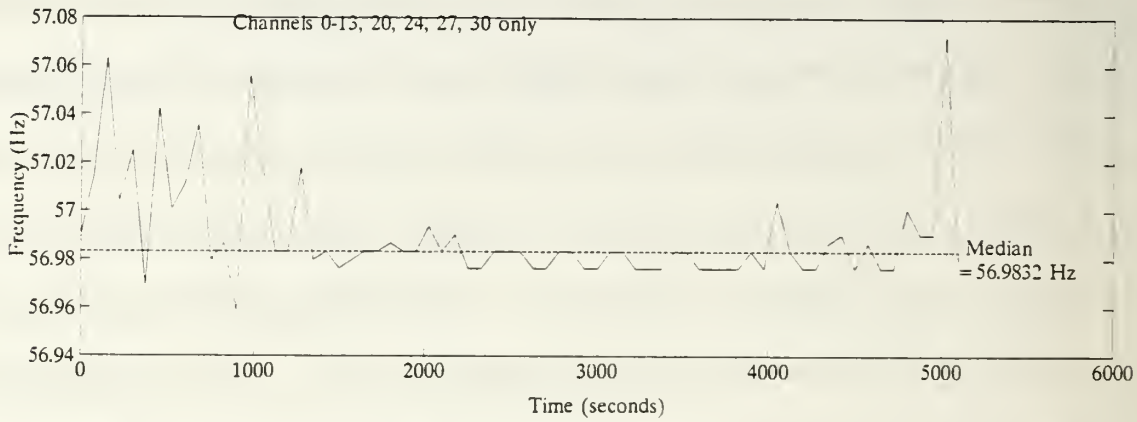

Figure 4.15: Median Frequency of 01270222 CW Signal.

3. Time Domain Processing

A pessimistic estimate of the time domain signal-to-noise ratio is made by finding the mean amplitude of all the points in a particular sequence, not trying to sort out signal from noise. The peak magnitude can then be divided by this value to obtain an estimate of the signal-to-noise ratio. However, if the post-processed signal has insufficient amplitude to be reliably detected in the noise field, the effectiveness of peak-picking is diminished. Unfortunately, this was the case for most m-sequences analyzed in this thesis. Although there were peaks in the data which were stationary in time

between several sequences, they rarely had sufficient amplitude or duration to distinguish them from similar features in the noise field. In addition, if multiple arrivals are present within the same sequence, the difficulty in discriminating the signal from noise will be increased. Note also that cross-channel average values cannot be used effectively in the time domain because of possible phase differences between hydrophones.

The 2047-digit m-sequence which was received at 01270005 GMT provided maximum gain and a relatively stable frequency. Figure 4.16(a and b) depicts the highest

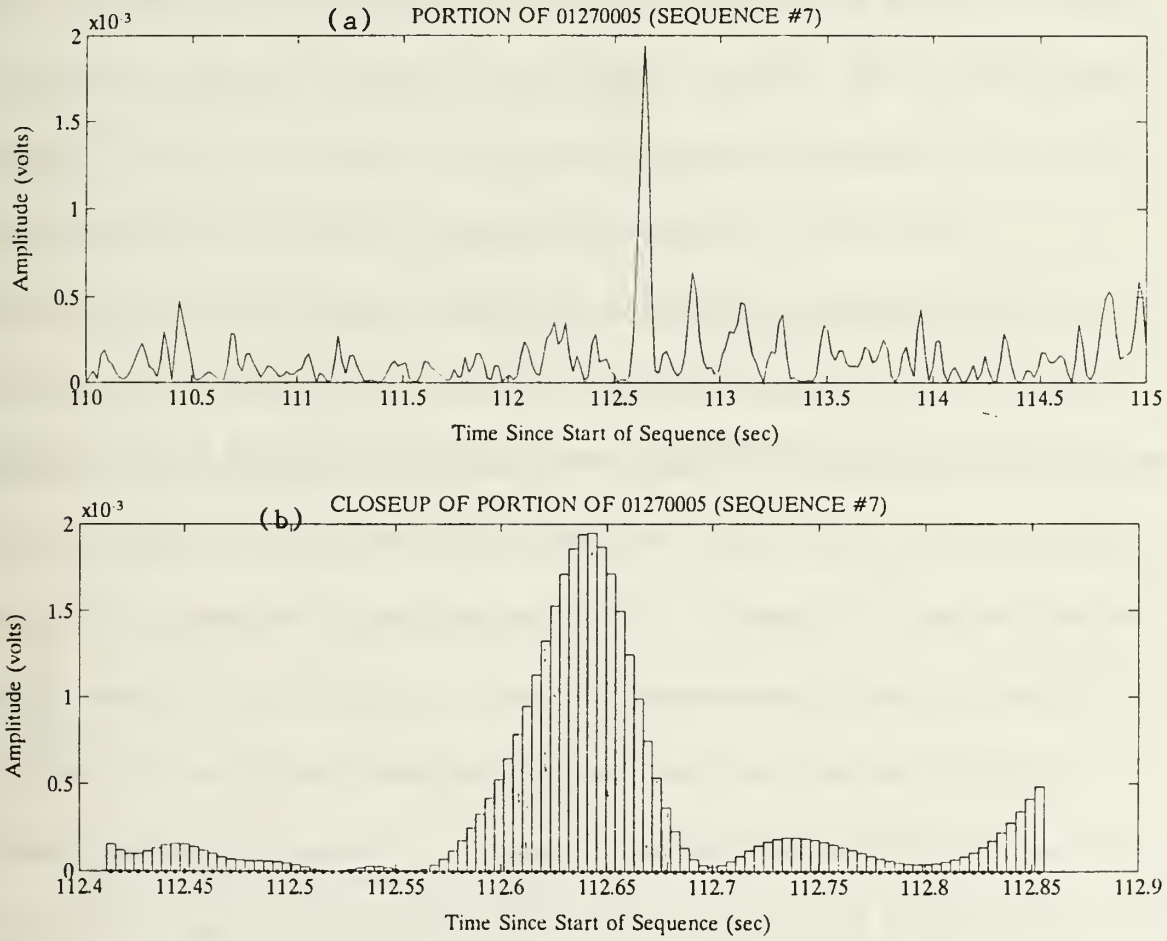
(a) PORTION OF 01270005 (SEQUENCE #7)
x10-3
Amplitude (volts)
Time Since Start of Sequence (sec)
(b) CLOSEUP OF PORTION OF 01270005 (SEQUENCE #7)
x10-3
Amplitude (volts)
Time Since Start of Sequence (sec)

Figure 4.16: Maximum Reasonable Peak of 01270005 2047-Digit M-Sequence.

peak detected in this reception that could be reasonably associated with the signal. The peak was 9 dB above the median noise level in the 180 second sequence and occurred at 112.6 seconds from the start of the 7th sequence. Figure 4.16(b) is a close-up view of the same peak, depicting the width of one m-sequence digit (.0877 seconds or 20 sample points). This particular peak, although higher than any others, was not repeated in other sequences at a corresponding time from the sequence start. Occurring only 2 hours before the 01270222CW signal, the 01270005 reception had similar noise levels. The coded signal level is 3 dB less than the CW level and thus the expected pre-processed SNR is approximately -30 dB. Since the maximum gain of the 2047 digit pulse compression is 33 dB, we can only expect a maximum output SNR of about 3 dB under ideal conditions.

If the maximum coherent integration time of the arrival exceeds the length of two consecutive sequences (359 seconds in this case), additional gain can be achieved by coherently summing adjacent sequences. In addition, more credence can be given to an arrival peak that is repetitive. Sequence averaging was performed in 3 minute increments from 6 minutes up to one hour in an effort to locate stable peaks. A peak was found in sequence number 8 of channel 5 at 169.25 seconds from the beginning of the sequence which also occurred in other sequences. Figure 4.17(a) and 4.17(b) illustrate this peak. Note that this peak not only has a smaller amplitude than Figure 4.16 but is also narrower, similar to the test data case in Figure 4.8. The peak was detectable above the noise in sequences 7, 8, 9, 10, and 13. Figure 4.18 shows the mean amplitude of this peak averaged over sequences 6 through 13. This mean peak is thus a coherent average between 8 sequences (about 24 minutes) and is roughly 6.7 dB above the mean noise.

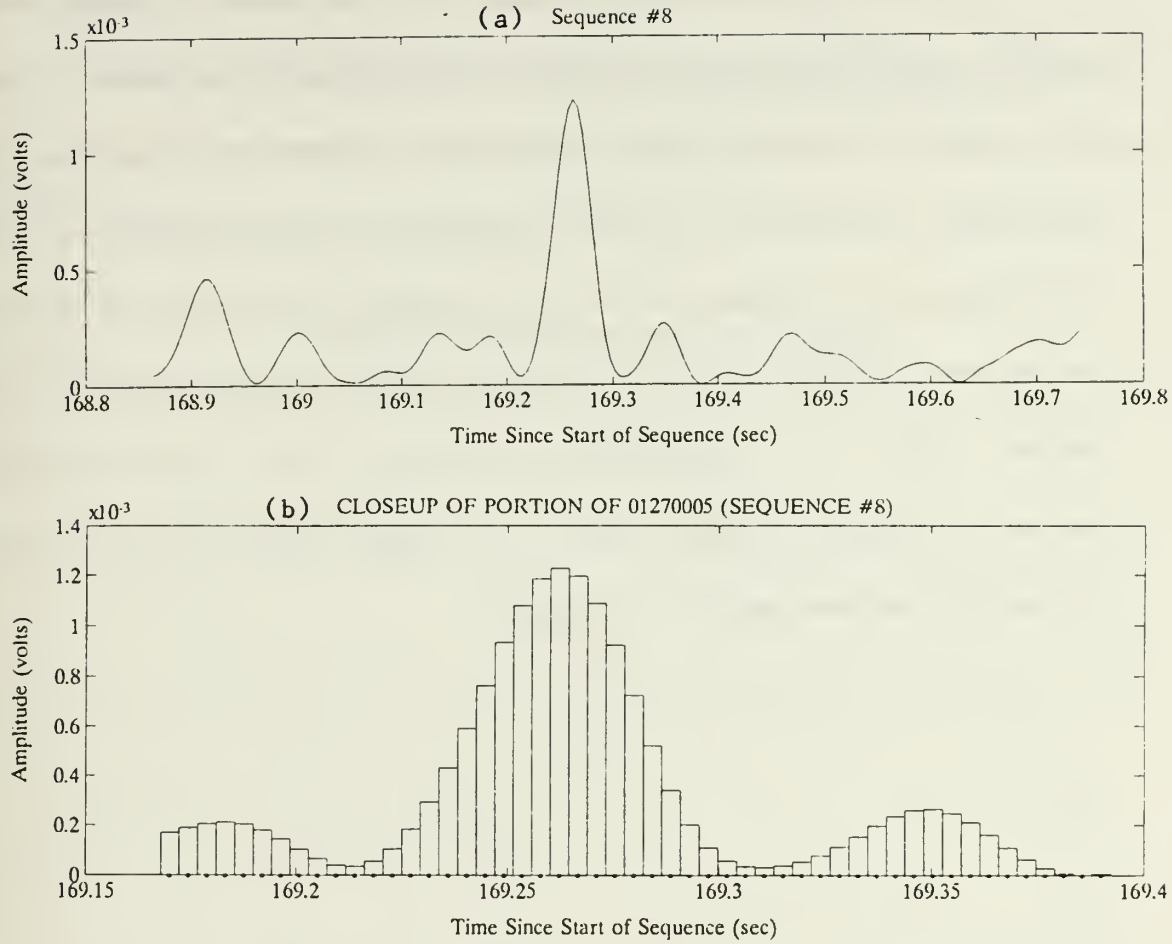

Figure 4.17: A Persistent Peak in Sequence 8 of 01270005 2047-Digit M-Sequence.

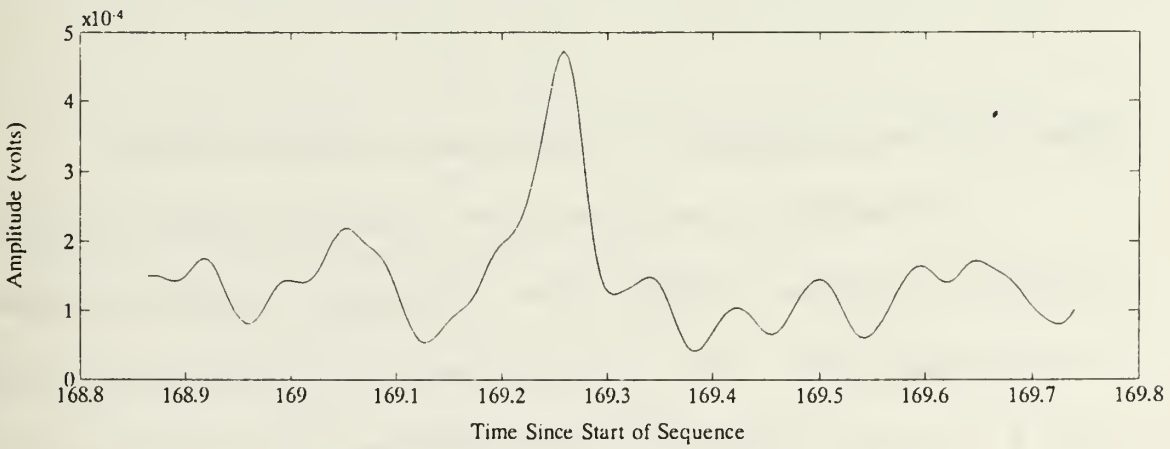

Figure 4.18: Mean Amplitude of a Persistent Peak in 01270005 2047-Digit M-Sequence.

Further evidence that the peak in Figures 4.17 and 4.18 is actually an arrival of 2047 m-sequence code is that they were not detected at all in sequences 1 through 6. The predicted first arrival of each sequence was 3 hours and 20 minutes after the transmission commenced due to acoustic propagation at a nominal 1500 m/s over 18 Mm. Since this particular recording commenced 3 hours and 5 minutes after the transmission, the first peak would be expected to occur 15 minutes into the recording after sequence number 5. Figure 4.19 is a plot of the mean level in a 0.0877 second band around the peak as a function of sequence number. The energy from the peak can be seen to increase after sequence number 5, as expected.

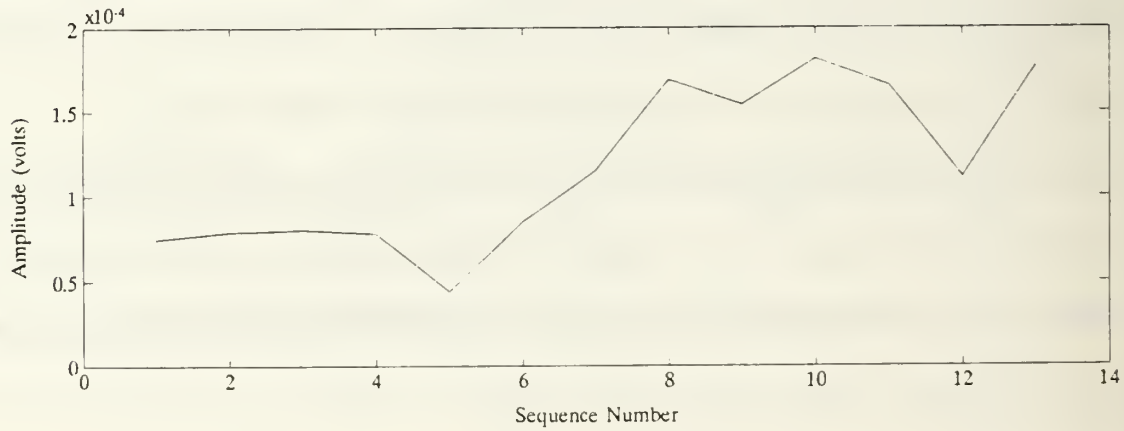

Figure 4.19: Mean Amplitude of 01270005 Peak vs Sequence Number.

The stability of an arrival determines its maximum coherent integration time since any time shift due to Doppler effects or dispersion, or amplitude decrease will diminish summing gains. A three dimensional perspective of the stability of the peak in Figure 4.17 is shown in Figure 4.20. Note that the peak varies in both amplitude as a function of sequence number and time position within any given sequence. In addition

to a myriad of oceanic causes, the variation could be produced or compounded by deviation of the received carrier frequency from the demodulation frequency, as discussed in Section 4.B.7. A gray-scale plot of this same reception is shown in Figure 4.21.

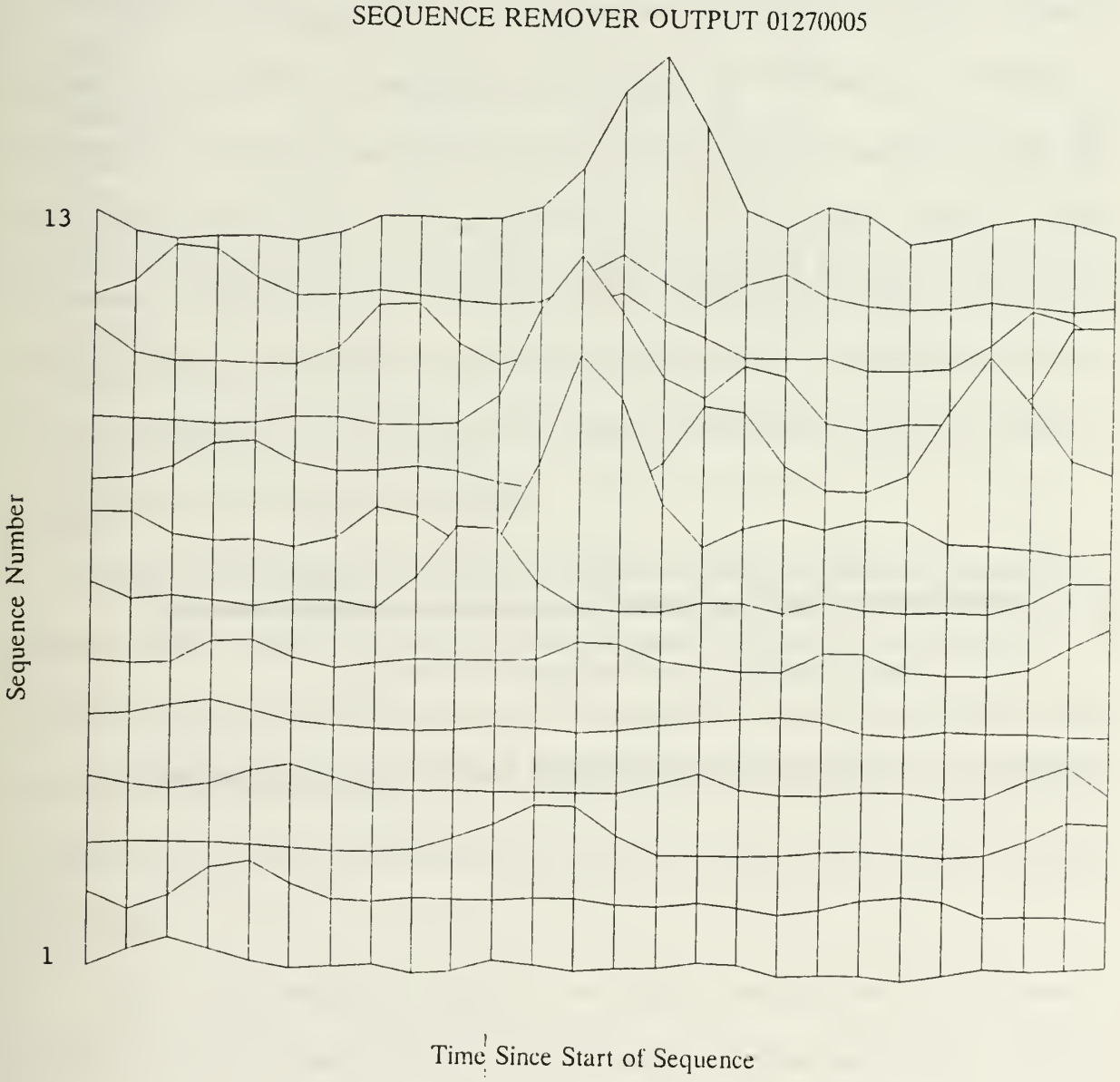

Figure 4.20: Variation of Amplitude and Position of 2047-Digit M-Sequence.

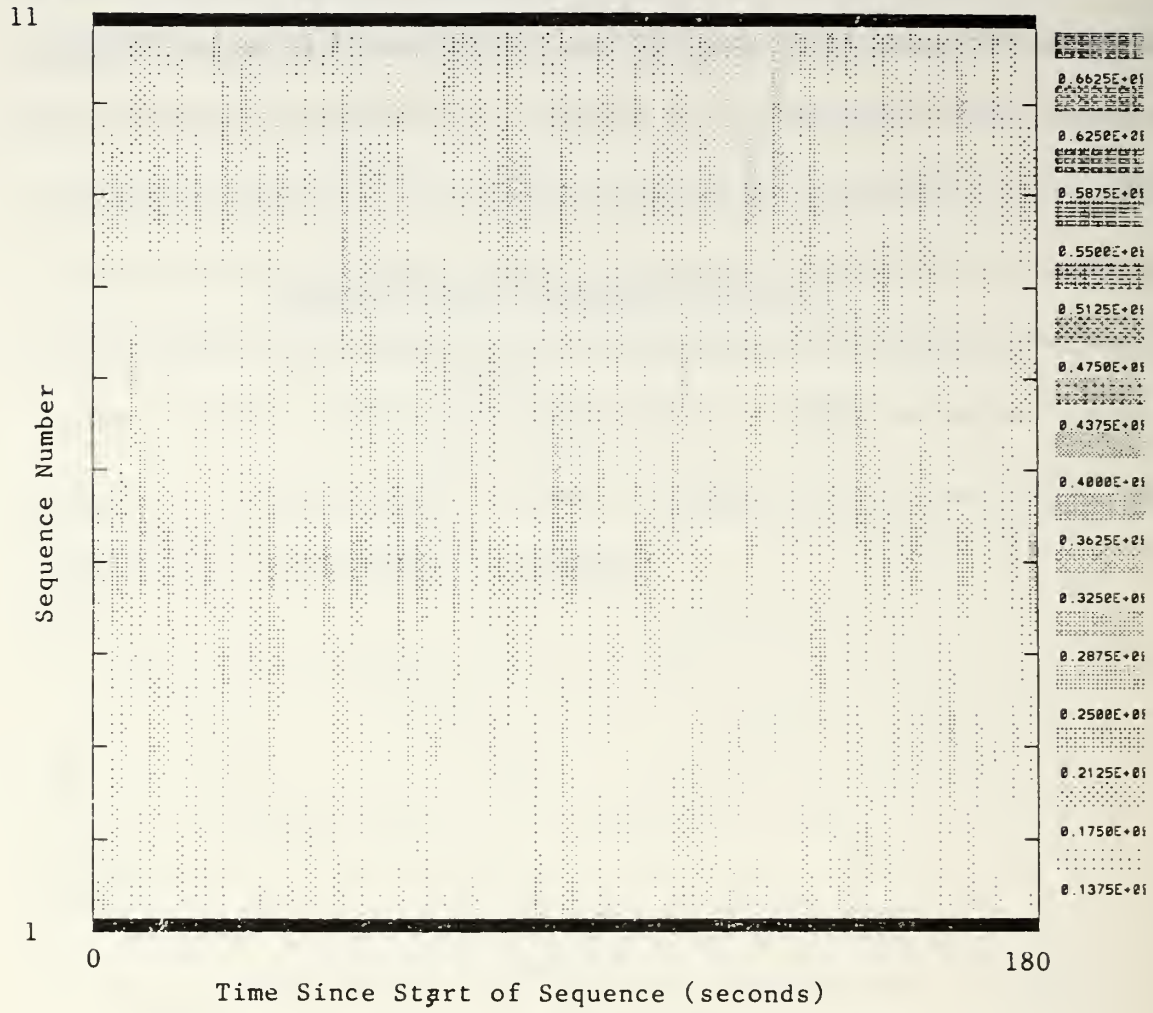

Figure 4.21: Gray-Scale Plot of 01270005 2047 M-Sequence (Channel 5).

V. CONCLUSIONS AND RECOMMENDATIONS

A. SUMMARY OF RESULTS

The Naval Postgraduate School in conjunction with The Monterey Bay Aquarium Institute, Woods Hole Oceanographic Institution, and Massachusetts Institute of Technology successfully participated in 1991 The Heard Island Feasibility Experiment, demonstrating the ability to receive global transmission of acoustic signals. The 18 Megameter transmission may have been the furthermost reception ever of an electronically generated acoustic signal in an oceanic medium. Over 1.5 gigabytes of data were collected from a 32 hydrophone array which intersected the deep sound channel in a horizontal zone of optimum reception.

The Naval Postgraduate School has gained a real-time processing capability for tomographic m-sequence signals using the Laboratory Workbench environment of the Concurrent 6400 data acquisition system. The potential to process phase-encoded signals as they are received will enhance the schools efforts in future acoustic tomography efforts. Adjustments to experimental parameters can now be made in a timely manner to optimize the result.

Only a very preliminary analysis of the Heard Island data has been performed in order to provide an overview of signal levels, integration times, and stability. Detection of the Heard Island was possible in both the frequency domain and the time domain. Spectral analysis of CW signals indicated a relatively stable carrier frequency (+/- about

0.01 Hz corresponding to about 0.5 kts of relative motion). The CW signal had a pre-processed level of about -28 +/- 3 dB which would imply an approximate -31 dB level for a coded signal in a similar noise field.

Time domain processing using pulse compression of the coded m-sequence enabled identification and resolution of occasional arrivals of the 2047-digit m-sequence. However the post-processed signal level was only about 9 dB for the maximum peak found and most arrival peaks had characteristics so similar to the noise field that reliable identification was not possible. The maximum coherent integration time found was 7 sequences or about 21 minutes of the 2047-digit sequence. The pre-processed levels were so low that small increases in the ambient noise level cause the post-processed SNR to fall below zero and prevent detection. Detection of the signal in both the frequency and time domains was intermittent as the signal often fell below the detection threshold.

B. CONCLUSIONS

1. Signal Processing

The signal processing software functions satisfactorily as designed but is sensitive to deviations of the carrier from the demodulation frequency, which occur as a result of relative motion between the source and receiver. A deviation of .01 Hz lowered the output amplitude by about 1.5 dB. A frequency deviation of more than 0.02 Hz is generally sufficient to make the signal undetectable in the time domain. Resampling of the signal may be necessary to minimize the effects of large Doppler frequency shifts

during moving ship tomography experiments with low SNR. The signal processing objectives of this thesis, as outlined in Chapter I have been fully met.

2. Heard Island Feasibility Experiment

Analysis of the Heard Island data has not been completed. The very cursory processing conducted in thesis was performed in order to verify the validity of the processing procedures and to begin to quantify gross characteristics of the data. Both frequency and time domain processing are viable analysis tools for this data, each possessing its own advantages.

The SNR calculations in Chapter IV using the median signal and noise levels indicate that the maximum processing gain of the longest sequence used (33 dB) is barely sufficient to make the signal detectable in the noise field. The average pre-processed SNR of -28 dB is indicative of a 19 dB acoustic power level rel 1μPa or in other words a 194 dB transmission loss. Indeed, the processed signal is tenuous, at best, and often undetectable in the noise field. Additional gain is required in order to reliably identify and analyze the arrival structure.

The maximum observed processed SNR in the time domain was 9 dB which corresponds to an arrival uncertainty of 29 milliseconds, using Eq. 3.1. This is approximately 3 times the maximum uncertainty objective of the Heard Island Feasibility Experiment. This maximum peak was not repetitive and was thus not reliably identifiable or conducive to averaging gain. Arrival peaks had pulse widths on the order of 0.1 seconds. Typical peaks had levels on the order of 4 dB which corresponds to an arrival time uncertainty of about 44 milliseconds. Therefore, without further processing gain, the

propagation from Heard Island to Monterey exhibits too much transmission loss to meet the experiment objectives.

C. RECOMMENDATIONS FOR ADDITIONAL WORK

There is much work left to do in analysis of the Heard Island data and improving the acoustic tomography capability at Naval Postgraduate School. Some recommendations in this regard are listed below:

1. Additional software development should be pursued to facilitate real-time Doppler frequency search and compensation. Automated phase correction would maximize the efficiency of the entire sequence removal process. The effects of relative motion are significant and must be removed in order to have persistent, high amplitude arrival peaks.

2. Continued efforts in reducing the noise in the recorded data would improve the output SNR. Pre-whitening of the noise using standard methods such as an auto-regressive filter may improve the performance of the noise filters as well as enhance coherent averaging. Also, the origin of the large noise transients that occurred in all channels should be investigated.

3. Array beamforming can be implemented to obtain up to 15 dB gain ideally for all 32 hydrophones. Up to 12 dB of gain could be achieved for the 18 functional hydrophones. In addition, modal beamforming would act as a modal filter and allow discrimination of the predominant modes.

4. Complete the initial survey of the data by creating matrices of the output data and identifying signal peaks. If sufficient gain can be realized, a detailed study of the arrival structure should be undertaken to begin to quantify the effects of dispersion and oceanic influences on the various propagation paths. Within the acoustic data from this experiment lies the signature of every oceanic process between Heard Island and Monterey for a period of 7 days. Further analysis could provide information on the variability of the marine environment over this period. Again, Doppler effects must be removed since they would mask any short term oceanographic effects on propagation.

5. The Heard Island Feasibility Experiment did not utilize precise timing between the source and the receivers because the objective was determining the accuracy of the arrival time and not the arrival time itself. For future tomography work, a precision

time synchronization system should be devised which could supplement the acoustic data with clocking information.

6. The volume of stored data in a tomography experiment can be enormous. However, since each digit is sampled numerous times, there is a redundancy in the data that can be reduced by decimation to conserve storage space. The C function fmet1.c has the ground work for a decimation scheme already installed in the form of a decimation factor that is menu-selectable. This feature should be expanded upon in future experiments.

7. Due to the oversampling described above, the top of the correlation peaks in the m-sequence are flat vice 'pointy' as would be the case for a single sample per digit. To more accurately resolve the actual arrival time, it would be advantageous to implement a method of locating a particular feature of the peak such as its center or forward edge. The center can be resolved by cross-correlating the peak with a square pulse of the same width which results in a 'pointed' peak.

Finally, since the propagation path from Heard Island to Monterey exhibits too much transmission loss to be useful for a long term study of global warming, it would be useful to conduct ray tracing between alternate sites in the Pacific basin. The Hamiltonian methods discussed in Chapter III could be used to determine the optimum locations of both a source and receiver to perform long term monitoring of global temperatures.

APPENDIX A

SCHEDULE OF RECORDED DATA

GREENWICH MEAN TIME	LOCAL TIME (PST)	SIGNAL TYPE	DATA LENGTH (MIN)	DATA SIZE (MBytes)	SAMPLE FREQ. (Hz)
01261525	01260725	CW	121	106	228 Hz
01261800	01261000	3/10	107	93	228 Hz
01262058	01261258	255 M	38	33	228 Hz
01270005	01261605	2047 M	73	64	228 Hz
01270222	01261922	CW	75	75	228 Hz
10270608	01262208	3/10	75	66	228 Hz
01271207	01270407	511 M	80	70	228 Hz
01271505	01270705	CW	153	138	228 Hz
01271807	01271007	3/10	153	134	228 Hz
01292115	01291315	255 M	138	241	456 Hz
01300011	01291611	1023 M	72	126	456 Hz
01312108	01311308	255 M	28	49	456 Hz
02010011	01311611	2047 M	18	31	456 Hz

Notes: 1. $GMT_{(received)} = GMT_{(tx)}$ + 3hrs & 20min

2. $PST_{(received)} = GMT_{(tx)}$ - 8 hrs + 3hrs & 20min

3. Acoustic Travel Time From Heard Island to Monterey is about 3 hours and 20 minutes.

Appendix B

SIGNAL PROCESSING PROGRAMS

The following programs are listed in this appendix:

A. RESAMP.C - Resampling for Doppler Correction and Clipping.

B. BANDPASS.C - 20^{th} order bandpass filter (45-69 Hz).

C. FDEMOD1.C - Demodulator.

D. LOPASS.C - 20^{th} order lowpass filter (0-12 Hz).

E. FMET1.C - Sequence Removal.

F. SPEWRIT.C - Writes spectral peak, frequency, and band noise files.

```
/*************************************************************************
RESAMP.C
This program forms a Lab Workbench module which resamples an input data
stream at exactly four times a menu specified carrier frequency.  It was
designed to be used with a buffer size of 10 seconds so the header file of
the data will have to be adjusted accordingly.  The program also
calculates the mean and standard deviation of each input buffer and will
clip all data in the buffer to a menu specified factor times the standard
deviation from the mean (default is 2).
*************************************************************************/
static char SccsID[] = "@(#)uftest.c8.1 (MASSCOMP) 2/5/90";
#include "lwbufunc.h"
#include "errors.h"
#include "math.h"
float  *idata, *odata;
double nfc=57.0, clip=2.0, pres[2280][32], past[2280][32], futr[2280][32];
int    bufcount, omsg=0,links=0, state=0,itype=0, otype=LWBFLOAT,
dimension=0, sizes[3]={0,0,0}, count=0, samples, rate=228.0, dx=0;
/*************************************************************************
 * >> Entry point from LWB — must be first function defined in file << *
 *************************************************************************/

uftest( action, inA, inB, inC, inD, outX, outY, debug )
int *action, *inA, *inB, *inC, *inD, outX, outY, *debug;
{
     int status;
     dx = *debug;
     switch ( *action )
     {
          case SETUPCMD:status = ufsetup( *inA, outX ); break;
          case RUNCMD:  status = ufrun( *inA, outX ); break;
          case STOPCMD:status = ufstop( *inA, outX ); break;
          case MENUCMD:status = ufmenu( *inA ); break;
          case DROPCMD:status = ufdrop(); break;
          default:      break;
     }
     return( status );
}
/************************** SETUP ************************/
ufsetup( in, out )
int in, out;
{
   int i;
   bufcount = 0;
/* set up menu label and initial value  */
     if ( !lwblabelmenu( 1, "Clip Factor" ) ) return( 0 );
     if ( !lwbupdatemenu( 1, "%f", clip) ) return( 0 );
     if ( !lwblabelmenu( 2, "Received Freq." ) ) return( 0 );
     if ( !lwbupdatemenu( 2, "%f", nfc) ) return( 0 );
/* get dimensionality and size of input message data */
     lwbgetparam( in, 0, LWBMSGDIM, &dimension );
```

```
    if (dx) printf( "resamp dimension=%d\n", dimension );
    lwbgetparam( in, 0, LWBMSGSIZES, sizes );
    if (dx) printf("resamp sizes=%d,%d,%d\n",sizes[0],sizes[1],sizes[2]);
    if ( dimension == 2 )
    {
        count = sizes[0] * sizes[1];
        samples = sizes[0];
     }
    else if ( dimension == 1 )
    {
        count = sizes[0];
        samples= sizes[0];
    }
    if ( dx ) printf( "resamp samples=%d\n", samples);
/* check input data type */
    lwbgetparam( in, 0, LWBMSGTYPE, &itype );
    if ( dx ) printf( "resamp itype=%d\n", itype );
    if ( itype != 5 )
    {
        lwberror( ERROR, "Input message data type must be float" );
        return( 0 );
    }
/* create a new message for output data */
    if ( !lwbnewmsg( &omsg ) ) return( 0 );
/* initialize message */
    if ( !lwbcopymsg( in, omsg ) ) return( 0 );
    if ( !lwbsamerange( in, omsg ) ) return( 0 );
    if ( !lwbsamedomain( in, omsg ) ) return( 0 );
/* create a data buffer for output message */
    if ( !lwbnewbuf( omsg ) ) return( 0 );
    if ( !lwbgetdatbuf( omsg, &odata ) ) return( 0 );
/* send message on its way and release the input */
    lwbwrite( omsg, out );
    lwbrelease( in );
    return( 1 );
}
/*************************  RUN  *******************************/
ufrun( in, out )
int in, out;
{
register int i,j,k;
int      mi,oi,ci,fi,pi;
double   rise, run, lower, slope, delt,otime,ntime,diff, mean[32],
         var[32], sigma[32], sum[32], num[32], max[32], min[32];
   /* sanity check */
    if ( !in || !omsg ) return( 0 );
    if ( !lwbgetparam( omsg, 0, LWBMSGLINKS, &links ) ) return( 0 );
    if ( links > 0 )
    {
        lwbunget( in );
        return( 1 );
```

```
    }
    if ( !lwbgetdatbuf( in, &idata ) ) return( 0 );
/*** IF THIS IS THE FIRST BUFFER ***/
   if(bufcount == 0)  /* Note: 1st 10 sec buffer output is bogus  */
   {
   /*** GET THE MEAN AND STD DEV FOR EACH CHANNEL OF THIS BUFFER ***/
      for ( j=0; j<32; j++)
      {
       sum[j] = 0;
        for(i=0; i<samples; i++)
         {
         k=j+(32*i);    /* index of idata[i][j]  */
          sum[j] = sum[j] + (double)idata[k];
         }
      }
      for(j=0; j<32; j++)
      {
        mean[j] = sum[j] / (double)samples;
         diff = 0.0;
         num[j] = 0.0;
        for(i=0; i<samples; i++)
         {
           k=j+(32*i);    /* index of idata[i][j]  */
            diff = (double)idata[k] - mean[j];
            num[j] = num[j] + (diff * diff);
         }
          var[j] = num[j] / (samples-1);
          sigma[j] = sqrt(var[j]);
           max[j] = mean[j] + (clip * sigma[j]);
          min[j] = mean[j] - (clip * sigma[j]);
      }

      /***  READ INPUT INTO PRESENT ARRAY & CLIP ***/
      for ( i=0; i<samples; i++)
      {
         for(j=0; j<32; j++)
         {
            k=j+(32*i);    /* index of idata[i][j]  */
            if((double)idata[k] < max[j] && (double)idata > min[j])
            {
             pres[i][j] = (double)idata[k];
            odata[k] =idata[k];
            }
            else      /* clip if input > clip * sigma */
            {
               if(idata[k] >= 0.0)
               {
                 pres[i][j]  = max[j];
                 odata[k] = (float)max[j];
               }
                if(idata[k] < 0.0)
```

```
                {
                  pres[i][j]  = min[j];
                  odata[k]=(float)min[j];
                }
              }
            }
          }

    }
/*** FOR ALL REMAINING BUFFERS ***/
    else
    {
       /***  READ INPUT INTO FUTURE ARRAY ***/
       for(i=0; i<samples; i++)
       {
         for(j=0; j<32; j++)
          {
           k=j+(32*i);    /* index of idata[i][j]  */
           futr[i][j] = (double)idata[k];
          }
       }
       /*** RESAMPLE THE 'PRESENT' ARRAY  ***/
       for ( i=0; i<samples; i++)
       {
          mi=(bufcount*samples)+i; /* running index for new time scale */
         ntime=(double)mi/(4.0*nfc);  /* new actual time for  current i */
         oi = ntime * rate;          /*old index just lower than newtime*/
          ci=oi-(bufcount*samples); /* current index lower than newtime */
         otime=(double)oi/rate;   /* old actual time lower than newtime */
          run = 1.0 / rate;      /* time difference between old samples */
         delt = ntime-otime;          /* time shift for this sample */
         for(j=0; j<32; j++)
         {
          k=j+(32*i);    /* index of odata[i][j]  */
           if (ci >= 0 && ci+1 <= samples)    /* both values in present */
           {
             lower = pres[ci][j];
             rise  = pres[ci+1][j] - lower;
           }
           else if(ci+1 == 0)                /* values split pres/past */
           {
             lower = past[samples-1][j];
             rise  = pres[0][j] - lower;
            }
           else if (ci < 0 && ci+1 == 0)    /* both values in past */
           {
            pi = samples+ ci;
             lower = past[pi][j];
             rise  = past[pi+1][j] - lower;
           }
           else if (ci == count-1)           /* values split pres/futr */
```

```
        {
           lower = pres[samples-1][j];
           rise  = futr[0][j] - lower;
        }
         else if (ci > samples-1)          /* both values in futr  */
         {
           fi = ci - samples;
            lower = futr[fi][j];
            rise  = futr[fi+1][j] - lower;
        }
         slope = rise / run;
        odata[k] = (float)(lower + (slope * delt));
        }
       }
/*** GET THE MEAN AND STD DEV FOR EACH CHANNEL OF THE OUTPUT  ***/
      for ( j=0; j<32; j++)
      {
          sum[j] = 0;
         for(i=0; i<samples; i++)
           k=j+(32*i);    /* index of odata[i][j]  */
         sum[j] = sum[j] + (double)odata[k];
      }
      for(j=0; j<32; j++)
      {
        mean[j] = sum[j] / (double)samples;
         diff = 0.0;
         num[j] = 0.0;
        for(i=0; i<samples; i++)
         {
          k=j+(32*i);    /* index of odata[i][j]  */
           diff = (double)odata[k] - mean[j];
           num[j] = num[j] + (diff * diff);
         }
        ave[j] = num[j] / (samples-1);
        sigma[j] = sqrt(var[j]);
         max[j] = mean[j] + (clip * sigma[j]);
        min[j] = mean[j] - (clip * sigma[j]);
      }
      /*  CLIP OUTPUT  */
      for(i=0; i<samples; i++)
      {
         for(j=0; j<32; j++)
          {
            k=j+(32*i);    /* index of odata[i][j]  */
            if((double)odata[k] > max[j] || (double)odata[k] < min[j])
            {
              if(odata[k] >= 0.0)
               odata[k] = (float)max[j];
              if(odata[k] < 0.0)
               odata[k] = (float)min[j];
            }
```

```
            }
        }
         /*** shift the data arrays toward the past  ***/
         for ( i=0; i<samples; i++)
         {
            for(j=0; j<32; j++)
             {
               past[i][j] = pres[i][j];
               pres[i][j] = futr[i][j];
             }
         }
    }
    bufcount++;
    lwbwrite( omsg, out );
    lwbrelease( in );
    return( 1 );
}

/************************  STOP  *******************************/
ufstop( in, out )
int in, out;
{
     if ( !in || !out ) return( 0 );
     lwbwrite( omsg, out );
     lwbrelease( in );
     return( 1 );
}
/************************  MENU  *******************************/
ufmenu( key )
int key;
{
   int i;
     if ( key == 1 )
     {
          lwbgetdouble( "Enter Clip Factor", 0.0, 500.0, &clip);
          if ( !lwbupdatemenu( 1, "%f", clip) ) return( 0 );
     }
     if ( key == 2 )
     {
          lwbgetdouble( "Enter Received Frequency", 0.0, 500.0, &nfc);
          if ( !lwbupdatemenu( 2, "%f", nfc) ) return( 0 );
     }
     return( 1 );
}
/************************  DROP  *******************************/
ufdrop()
{
     if ( !lwbfreemsg( &omsg ) ) return( 0 );
     return( 1 );
}
```

```
/*************************************************************************
BANDPASS.c
   This function implements a 20th order Finite Impulse Response (FIR)
   bandpass filter where the filter coefficients were computed using
   the FIR1 routine in MATLAB.  If W(n) denotes a window, where 1<n<N,
   and the impulse response of the ideal filter is h(n), where h(n) is
   the inverse Fourier transform of the ideal frequency response, then
   the windowed digital filter coefficients are given by
   b(n)= W(n)h(n), 1<n<N.  The user can select either 228 or 456 hz sample
   rate as appropriate for the Heard Island Data.
*************************************************************************/
static char SccsID[] = "@(#)uftest.c8.1 (MASSCOMP) 2/5/90";
#include "lwbufunc.h"
#include "errors.h"
float *idata, *odata;
int  omsg=0, links=0, state=0, itype=0, otype=LWBFLOAT, dimension=0,
     sizes[3]={0,0,0},  count=0, rate=228, dx=0;
double b_228[21] = {9.67912035115705e-04,  7.59486633167777e-19,
                    7.34078321026075e-03, -8.92845210001374e-18,
                   -4.46377512008602e-02,  2.42224318618152e-17,
                    1.21524732772206e-01, -3.05358871759598e-17,
                   -2.05919147191579e-01,  1.42856914119550e-17,
                    2.43090995320418e-01,  1.42856914119550e-17,
                   -2.05919147191579e-01, -3.05358871759598e-17,
                    1.21524732772206e-01,  2.42224318618152e-17,
                   -4.46377512008602e-02, -8.92845210001374e-18,
                    7.34078321026076e-03,  7.59486633167777e-19,
                    9.67912035115705e-04},
       b_456[21] = {3.01389808517666e-07,  9.65884999452910e-03,
                    2.44744230763637e-02,  3.00114545478715e-02,
                   -1.25908275156001e-06, -6.76084123580341e-02,
                   -1.26039643870063e-01, -1.09389973461736e-01,
                    1.11960800803549e-06,  1.36893131470553e-01,
                    1.98959837606674e-01,  1.36893131470553e-01,
                    1.11960800803549e-06, -1.09389973461736e-01,
                   -1.26039643870063e-01, -6.76084123580341e-02,
                   -1.25908275156001e-06,  3.00114545478715e-02,
                    2.44744230763637e-02,  9.65884999452910e-03,
                    3.01389808517666e-07},
    stor[20]={1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
     B[21], otemp[500];
/*************************************************************************
 * >> Entry point from LWB — must be first function defined in file << *
 *************************************************************************/
uftest( action, inA, inB, inC, inD, outX, outY, debug )
int *action, *inA, *inB, *inC, *inD, outX, outY, *debug;
{
     int status;
     dx = *debug;
     switch ( *action )
     {
```

```
            case SETUPCMD:status = ufsetup( *inA, outX ); break;
            case RUNCMD:  status = ufrun( *inA, outX ); break;
            case STOPCMD:status = ufstop( *inA, outX ); break;
            case MENUCMD:status = ufmenu( *inA ); break;
            case DROPCMD:status = ufdrop(); break;
            default:      break;
        }
        return( status );
}
ufsetup( in, out )
int in, out;
{
 int i;
/* set up menu label and initial value  */
        if ( !lwblabelmenu( 1, "Sample Rate" ) ) return( 0 );
        if ( !lwbupdatemenu( 1, "%d", rate) ) return( 0 );
        if (rate==228)
           for(i=0; i < 21; i++)  B[i] = b_228[i];
        if (rate==456)
           for(i=0; i < 21; i++)  B[i] = b_456[i];
/* get dimensionality and size of input message data */
        lwbgetparam( in, 0, LWBMSGDIM, &dimension );
        if ( dx ) printf( "dimension=%d\n", dimension );
        lwbgetparam( in, 0, LWBMSGSIZES, sizes );
        if ( dx ) printf( "sizes=%d,%d,%d\n", sizes[0], sizes[1], sizes[2] );
        if ( dimension == 2 )
             count = sizes[0] * sizes[1];
        else if ( dimension == 1 )
             count = sizes[0];
        if ( dx ) printf( "count=%d\n", count );
/* check input data type */
        lwbgetparam( in, 0, LWBMSGTYPE, &itype );
        if ( dx ) printf( "itype=%d\n", itype );
        if ( itype != 5 )
        {
             lwberror( ERROR, "Input message data type must be float" );
             return( 0 );
        }
/* create a new message for output data */
        if ( !lwbnewmsg( &omsg ) ) return( 0 );
/* initialize message */
        if ( !lwbcopymsg( in, omsg ) ) return( 0 );
        if ( !lwbsamerange( in, omsg ) ) return( 0 );
        if ( !lwbsamedomain( in, omsg ) ) return( 0 );
/* create a data buffer for output message */
        if ( !lwbnewbuf( omsg ) ) return( 0 );
        if ( !lwbgetdatbuf( omsg, &odata ) ) return( 0 );
/* send message on its way and release the input */
        lwbwrite( omsg, out );
        lwbrelease( in );
        return( 1 );
```

```
}
/************************* RUN ***************************************/
ufrun( in, out )
int in, out;
{
register int i, j, k;
/* sanity check */
     if ( !in || !omsg ) return( 0 );
     if ( !lwbgetparam( omsg, 0, LWBMSGLINKS, &links ) ) return( 0 );
     if ( links > 0 )
     {
          lwbunget( in );
          return( 1 );
     }
     if ( !lwbgetdatbuf( in, &idata ) ) return( 0 );
/**** PROCESS THE DATA ***/
     for ( i = 0; i < 20; i++)  /* process first 20 data points in buffer
*/
        {
           otemp[i]=0.0;                    /* initialize otemp[i] */
              for(j=0; j < i+1; j++)
                otemp[i] = otemp[i]+B[j] * (double)idata[i-j];
              for(j=i+1; j < 21; j++)
                otemp[i] = otemp[i]+B[j] * stor[20-j+i];
          odata[i] = (float)(otemp[i]);
        }
     for ( i = 20; i < count; i++)  /* process remainder of buffer */
     {
          otemp[i]=0.0;                /* initialize otemp[i] */
          for(j=0; j < 21; j++)
                otemp[i] = otemp[i]+B[j]*(double)idata[i-j];
         odata[i] = (float)(otemp[i]);
     }
     for ( i = 0; i < 20; i++)  /* store values for next buffer  */
        stor[i] = (double)idata[count-20+i];
     lwbwrite( omsg, out );
     lwbrelease( in );
     return( 1 );
}
/************************  STOP  ************************************/
ufstop( in, out )
int in, out;
{
     if ( !in || !out ) return( 0 );
     lwbwrite( omsg, out );
     lwbrelease( in );
     return( 1 );
}
/************************  MENU  ************************************/
ufmenu( key )
int key;
```

```
{
   int i;
     if ( key == 1 )
     {
          lwbgetint( "Enter Sample Rate(228 or 456)", 228, 456, &rate);
          if ( !lwbupdatemenu( 1, "%d", rate) ) return( 0 );
     }
     if (rate==228)
        for(i=0; i < 21; i++)   B[i] = b_228[i];
     if (rate==456)
        for(i=0; i < 21; i++)   B[i] = b_456[i];
     return( 1 );
}
/************************** DROP **********************************/
ufdrop()
{
     if ( !lwbfreemsg( &omsg ) ) return( 0 );
     return( 1 );
}
```

```
/*****************************************************************************
LOPASS.c
This is an implementation of a 20th order lopass filter whose coefficients
are produced by matlab "fir1" function.
*****************************************************************************/
static char SccsID[] = "@(#)uftest.c8.1 (MASSCOMP) 2/5/90";
#include "lwbufunc.h"
#include "errors.h"
/* input and output data pointers */
float *idata, *odata;
int  omsg=0, links=0, state=0, itype=0, otype=LWBFLOAT, dimension=0,
     sizes[3]={0,0,0},  count=0, rate=228, dx=0;
double b_228[21] = {-5.51905550574465e-04, 7.85822093157633e-04,
                4.18562857927650e-03,     1.18776604147200e-02,
                2.54520047550161e-02,     4.51129392503578e-02,
                6.92922548764514e-02,     9.47857101643358e-02,
                1.17413185001872e-01,     1.33031630503429e-01,
                1.38608237639420e-01,     1.33031630503429e-01,
                1.17413185001872e-01,     9.47857101643358e-02,
                6.92922548764514e-02,     4.51129392503578e-02,
                2.54520047550161e-02,     1.18776604147200e-02,
                4.18562857927650e-03,     7.85822093157633e-04,
               -5.51905550574465e-04}
     b_456[21] = { 5.02260174274603e-03,   7.15115484736450e-03,
                1.28132371720883e-02,     2.22206445774868e-02,
                3.49701195466541e-02,     5.00555223943375e-02,
                6.59856590758503e-02,     8.09913198417360e-02,
                9.32883970105506e-02,     1.01352993037039e-01,
                1.04161357873986e-01,     1.01352993037039e-01,
                9.32883970105506e-02,     8.09913198417360e-02,
                6.59856590758504e-02,     5.00555223943375e-02,
                3.49701195466541e-02,     2.22206445774868e-02,
                1.28132371720883e-02,     7.15115484736450e-03,
                5.02260174274603e-03}
    stor[20]={1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
     B[21], otemp[500];

/*****************************************************************************
 * >> Entry point from LWB -- must be first function defined in file << *
 *****************************************************************************/
uftest( action, inA, inB, inC, inD, outX, outY, debug )
int *action, *inA, *inB, *inC, *inD, outX, outY, *debug;
{
     int status;
     dx = *debug;
     switch ( *action )
     {
          case SETUPCMD:status = ufsetup( *inA, outX ); break;
          case RUNCMD:  status = ufrun( *inA, outX ); break;
          case STOPCMD:status = ufstop( *inA, outX ); break;
          case MENUCMD:status = ufmenu( *inA ); break;
```

```
        case DROPCMD:status = ufdrop(); break;
        default:      break;
    }
    return( status );
}
ufsetup( in, out )
int in, out;
{
       int i;
/* set up menu label and initial value  */
    if ( !lwblabelmenu( 1, "Sample Rate" ) ) return( 0 );
    if ( !lwbupdatemenu( 1, "%d", rate) ) return( 0 );
    if (rate==228)
       for(i=0; i < 21; i++)  B[i] = b_228[i];
    if (rate==456)
       for(i=0; i < 21; i++)  B[i] = b_456[i];
/* get dimensionality and size of input message data */
    lwbgetparam( in, 0, LWBMSGDIM, &dimension );
    if ( dx ) printf( "dimension=%d\n", dimension );
    lwbgetparam( in, 0, LWBMSGSIZES, sizes );
    if ( dx ) printf( "sizes=%d,%d,%d\n", sizes[0], sizes[1], sizes[2] );
    if ( dimension == 2 )      count = sizes[0] * sizes[1];
    else if ( dimension == 1 )count = sizes[0];
    if ( dx ) printf( "count=%d\n", count );
/* check input data type */
    lwbgetparam( in, 0, LWBMSGTYPE, &itype );
    if ( dx ) printf( "itype=%d\n", itype );
    if ( itype != 5 )
    {
        lwberror( ERROR, "Input message data type must be float" );
        return( 0 );
    }
/* create a new message for output data */
    if ( !lwbnewmsg( &omsg ) ) return( 0 );
/* initialize message */
    if ( !lwbcopymsg( in, omsg ) ) return( 0 );
    if ( !lwbsamerange( in, omsg ) ) return( 0 );
    if ( !lwbsamedomain( in, omsg ) ) return( 0 );
/* create a data buffer for output message */
    if ( !lwbnewbuf( omsg ) ) return( 0 );
    if ( !lwbgetdatbuf( omsg, &odata ) ) return( 0 );
/* send message on its way and release the input */
    lwbwrite( omsg, out );
    lwbrelease( in );
    return( 1 );
}
/************************* RUN ****************************************/
ufrun( in, out )
int in, out;
{
register int i, j, k;
```

```
/* sanity check */
     if ( !in || !omsg ) return( 0 );
     if ( !lwbgetparam( omsg, 0, LWBMSGLINKS, &links ) ) return( 0 );
     if ( links > 0 )
     {
          lwbunget( in );
          return( 1 );
     }
     if ( !lwbgetdatbuf( in, &idata ) ) return( 0 );
/**** PROCESS THE DATA ***/
     for ( i = 0; i < 20; i++)  /* process first 20 data points in buffer
*/
      {
           otemp[i]=0.0;                    /* initialize otemp[i] */
             for(j=0; j < i+1; j++)
           otemp[i] = otemp[i]+B[j] * (double)idata[i-j];
             for(j=i+1; j < 21; j++)
               otemp[i] = otemp[i]+B[j] * stor[20-j+i];
            odata[i] = (float)(otemp[i]);
     }
     for ( i = 20; i < count; i++)  /* process remainder of buffer */
     {
          otemp[i]=0.0;                /* initialize otemp[i] */
          for(j=0; j < 21; j++)
                otemp[i] = otemp[i]+B[j]*(double)idata[i-j];
         odata[i] = (float)(otemp[i]);
     }
     for ( i = 0; i < 20; i++)  /* store values for next buffer  */
        stor[i] = (double)idata[count-20+i];
     lwbwrite( omsg, out );
     lwbrelease( in );
     return( 1 );
}
/************************  STOP  ************************************/
ufstop( in, out )
int in, out;
{
     if ( !in || !out ) return( 0 );
     lwbwrite( omsg, out );
     lwbrelease( in );
     return( 1 );
}
/************************  MENU  ************************************/
ufmenu( key )
int key;
{
   int i;
     if ( key == 1 )
     {
          lwbgetint( "Enter Sample Rate(228 or 456)", 228, 456, &rate);
          if ( !lwbupdatemenu( 1, "%d", rate) ) return( 0 );
```

```
    }
    if (rate==228)
       for(i=0; i < 21; i++)   B[i] = b_228[i];
    if (rate==456)
       for(i=0; i < 21; i++)   B[i] = b_456[i];
    return( 1 );
}
/*************************** DROP ***********************************/
ufdrop()
{
    if ( !lwbfreemsg( &omsg ) ) return( 0 );
    return( 1 );
}
```

```
/*************************************************************************
FDEMOD1.c
This program forms a user function in Lab Workbench to demodulate the
input data stream and send two output data streams at x and y consisting
to the real and imaginary parts, respectively.  The user can select the
carrier and sample frequency via menu.  Note:  The data MUST be
demodulated at exactly the frequency of the received carrier frequency for
the sequence removal to function optimally.
*************************************************************************/
static char SccsID[] = "@(#)uftest.c8.1 (MASSCOMP) 2/5/90";
#include "lwbufunc.h"
#include "errors.h"
#include <math.h>
/* input and output data pointers */
float   *idata, *odataX, *odataY;
int omsgX=0,  omsgY=0,  linksX=0,  linksY=0,  itype=0,  idimension=0,
odimension=1,
     insizes[3]={0,0,0}, outsizes[3], count=0,dx=0;
static  n=0;
double  fc=56.983, fs=228.0, theta=0.0, pi, doppler=0.0, doppler_bin=0.0,
        snd_vel=1500.0;
/*************************************************************************
 * >> Entry point from LWB -- must be first function defined in file << *
 *************************************************************************/
uftest( action, inA, inB, inC, inD, outX, outY, debug )
int *action, *inA, *inB, *inC, *inD, outX, outY, *debug;
{
     int status;
     dx = *debug;
     switch ( *action )
     {
          case SETUPCMD:status = ufsetup( *inA, outX, outY ); break;
          case RUNCMD:  status = ufrun( *inA, outX, outY ); break;
          case STOPCMD:status = ufstop( *inA, outX, outY ); break;
          case MENUCMD:status = ufmenu( *inA ); break;
          case DROPCMD:status = ufdrop(); break;
          default:      break;
     }
     return( status );
}
/*******************************  SETUP ********************************/
ufsetup( in, outX, outY )
int in, outX, outY;
{
double asin();
n=0;
odimension=1;
pi = 2 * asin(1.0);
/* set up menu label and initial value */
       if ( !lwblabelmenu( 1, "Carrier Frequency" ) ) return( 0 );
       if ( !lwbupdatemenu( 1, "%f", fc ) ) return( 0 );
```

```
        if ( !lwblabelmenu( 2, "Sample Frequency" ) ) return( 0 );
        if ( !lwbupdatemenu( 2, "%f", fs ) ) return( 0 );
        if ( !lwblabelmenu( 3, "Doppler (m/s)" ) ) return( 0 );
        if ( !lwbupdatemenu( 3, "%f", doppler ) ) return( 0 );
        if ( !lwblabelmenu( 4, "Sound Speed (m/s)" ) ) return( 0 );
        if ( !lwbupdatemenu( 4, "%f", snd_vel ) ) return( 0 );
/* compute theta */
        theta = 2 * pi * ( fc / fs );
        if ( dx ) printf( "demod theta from setup =%f\n", theta );
/* compute doppler correction */
        doppler_bin = pi * doppler / (2 * snd_vel);
        if ( dx ) printf( "demod doppler_bin from setup =%f\n",
doppler_bin );
/* check input messages */
     /* get dimensionality of input message data */
     lwbgetparam( in, 0, LWBMSGDIM, &idimension );
     if ( dx ) printf( "demod idimension=%d\n", idimension );
     /* get size of input message data */
     lwbgetparam( in, 0, LWBMSGSIZES, insizes );
     count = insizes[0];
     if ( dx ) printf( "demod count=%d\n", count );
     /* check input data type */
     lwbgetparam( in, 0, LWBMSGTYPE, &itype );
     if ( dx ) printf( "demod itype=%d\n", itype );
     if ( itype != 5 )
     {
          lwberror( ERROR, "Input message data type must be FLOAT" );
          return( 0 );
     }
/* create a new message for output data */
     if ( !lwbnewmsg( &omsgX ) ) return( 0 );
        if ( !lwbnewmsg( &omsgY ) ) return( 0 );
     /* copy msg header info from input to output message */
     if ( !lwbcopymsg( in, omsgX ) ) return( 0 );
        if ( !lwbcopymsg( in, omsgY ) ) return( 0 );
     outsizes[0]=insizes[0];
     outsizes[1]=1;
     outsizes[2]=0;
      if ( !lwbsetparam( omsgX, 0, LWBMSGSIZES, outsizes ) ) return( 0 );
      if ( !lwbsetparam( omsgY, 0, LWBMSGSIZES, outsizes ) ) return( 0 );
      if ( !lwbsetparam( omsgX, 0, LWBMSGDIM, &odimension ) ) return( 0 );
      if ( !lwbsetparam( omsgY, 0, LWBMSGDIM, &odimension ) ) return( 0 );
     /* set output range parameters */
      if ( !lwbnewrange( omsgX, 1 ) ) return( 0 );
      if ( !lwbnewrange( omsgY, 1 ) ) return( 0 );
     if ( !lwbcopyrange( in, omsgX ) ) return( 0 );
      if ( !lwbcopyrange( in, omsgY ) ) return( 0 );
      if ( !lwbnewdomain( omsgX ) ) return( 0 );
      if ( !lwbnewdomain( omsgY ) ) return( 0 );
     if ( !lwbcopydomain( in, omsgX ) ) return( 0 );
      if ( !lwbcopydomain( in, omsgY ) ) return( 0 );
```

```
     /* create a data buffer for output message */
     if ( !lwbnewbuf( omsgX ) ) return( 0 );
      if ( !lwbnewbuf( omsgY ) ) return( 0 );
     if ( !lwbgetdatbuf( omsgX, &odataX ) ) return( 0 );
      if ( !lwbgetdatbuf( omsgY, &odataY ) ) return( 0 );
     lwbgetparam( omsgX, 0, LWBMSGDIM, &odimension );
     if ( dx ) printf( "demod odimension=%d\n", odimension );
/* send message on its way */
     lwbwrite( omsgX, outX );
      lwbwrite( omsgY, outY );
/* and release the input */
     lwbrelease( in );
     return( 1 );
}
/*************************** RUN ***********************************/
ufrun( in, outX, outY )
int in, outX, outY;
{
register int i, j;
static       m;
double       temp, cos(), sin();
/* sanity check */
     if ( !in || !omsgX || !omsgY ) return( 0 );
     if ( !lwbgetparam( omsgX, 0, LWBMSGLINKS, &linksX ) ) return( 0 );
      if ( !lwbgetparam( omsgY, 0, LWBMSGLINKS, &linksY ) ) return( 0 );
     if ( (linksX > 0) || (linksY > 0) )
     {
          lwbunget( in );
          return( 1 );
     }
     if ( !lwbgetdatbuf( in, &idata ) ) return( 0 );
     for (i=0; i < count; i++)
     {
          temp = (double)idata[i];
          odataX[i] =  (float)(temp * cos(n*theta));
          odataY[i] =  (float)(temp * sin(n*theta));
            n=n+1;
     }
     m++;
     lwbwrite( omsgX, outX );
      lwbwrite( omsgY, outY );
     lwbrelease( in );
     return( 1 );
}
/******************************* STOP **********************************/
ufstop( in, outX, outY )
int in, outX, outY;
{
     if ( !in || !outX || !outY ) return( 0 );
     lwbwrite( omsgX, outX );
      lwbwrite( omsgY, outY );
```

```
    lwbrelease( in );
    return( 1 );
}

/****************************** MENU ********************************/
ufmenu( key )
int key;
{
    if ( key == 1 )
    {
        lwbgetdouble( "Enter Carrier Frequency", 1.0, 1000.0, &fc );
        if ( !lwbupdatemenu( 1, "%f", (float)fc ) ) return( 0 );
    }
    if ( key == 2 )
       {
            lwbgetdouble( "Enter Sample Frequency", 1.0, 1000.0, &fs
);
            if ( !lwbupdatemenu( 2, "%f", (float)fs ) ) return( 0 );
       }
   if ( key == 3 )
       {
            lwbgetdouble( "Enter Total Doppler (m/s)",-5.0,5.0,
&doppler);
            if ( !lwbupdatemenu( 3, "%f", (float)doppler) ) return( 0
);
       }

   if ( key == 4 )
       {
              lwbgetdouble("Enter Sound Velocity
(m/s)",1400.0,1500.0,&snd_vel);
          if ( !lwbupdatemenu( 4, "%f", (float)snd_vel) ) return( 0 );
       }
     theta = 2 * pi * ( fc / fs );
     doppler_bin = pi * doppler / (2 * snd_vel);
    return( 1 );
}

/****************************** DROP ********************************/
ufdrop()
{
    if ( !lwbfreemsg( &omsgX ) ) return( 0 );
       if ( !lwbfreemsg( &omsgY ) ) return( 0 );
    return( 1 );
}
```

```
/*************************************************************************
FMET1.c
This is the sequence removal program which calls a subroutine developed by
Metzger to do the pulse compression.  This program forms a user function
in Lab Workbench and allows menu selection of the sample freq. and law of
the code.  Note that LWB doesn't let you enter octal so enter the decimal
law of the code and it will be converted and displayed as the octal law on
the menu.
*************************************************************************/
static char SccsID[] = "@(#)uftest.c8.1 (MASSCOMP) 2/5/90";
#include "lwbufunc.h"
#include "errors.h"
#include <malloc.h>
#include <stdio.h>
#include <math.h>
#define  PI 3.1415926536

/* input and output data pointers */
float     *indataA, *indataB, *outdata;
double    iangle=45.0, angle, fact_r, fact_i;

float     slope,
          baseline,
           dfs=456,           /* sample freq (changable by menu) */
          dfc=57,            /* carrier freq */
           cycles_dig=5,      /* carrier cycles per digit */
           thresh_fact=50;   /* printout if output=thresh_fact*out_ave */

/* (global) variables (put these in a macrofile later) */
int    law=351,             /* decimal law..program converts to octal */
      initial=1,           /* initial register load */
       counts,             /* data points in one input buffer */
       dem_dig=20,         /* data points in one phase digit  */
       seq_len=1,          /* data points in one m-sequence..computed */
      degree,             /* number of registers in sequence generator */
       outmsg=0,
       links=0,
       intypeA=0,          /* LWB inputA data type (ie float=5) */
       intypeB=0,
       indimA=0,           /* dimension of inputA buffer array */
       indimB=0,
       insizesA[3],        /* array to hold input buffer size data */
       insizesB[3],
       osizes[3],
      gain,
      conv_gain,          /* value of least sig. bit of data */
       step=1,             /* decimation in time factor if used */
       dx=0,               /* flag to execute debug statements */
      scram[2047],        /* array to hold scrambled input data */
      unscram[2047],      /* array to hold unscrambled input data */
      dec_data[83000],    /* array to accumulate input data */
```

```
       seq_sum[80],        /* array to hold the 0th item of each sequence
                              passed to hadamard routine so bias can be
                              accumulated */
       cbuf[4100];         /* array to temporarily hold one seq_len of
                              data to be passed to hadamard routine */
static long k=0,             /*  data point counter  */
            l=0,            /*  buffer counter  */
            mark_1,         /* marks first excess data in k array */
            num_wrap;       /* number of exess data points in k array */

/*************************************************************************
 * >> Entry point from LWB — must be first function defined in file << *
 *************************************************************************/

uftest( action, inA, inB, inC, inD, outX, outY, debug )
int *action, *inA, *inB, *inC, *inD, outX, outY, *debug;
{
     int status;
     dx = *debug;
     switch ( *action )
     {
          case SETUPCMD:status = ufsetup( *inA, *inB, outX ); break;
          case RUNCMD:  status = ufrun( *inA, *inB, outX ); break;
          case STOPCMD:status = ufstop( *inA, *inB, outX ); break;
          case MENUCMD:status = ufmenu( *inA, *inB ); break;
          case DROPCMD:status = ufdrop(); break;
          default:      break;
     }
     return( status );
}

/********************************  SETUP  ****************************/
ufsetup( inA, inB, out )
int inA, inB, out;
{
int          temp=law, rev_law=0, end_bit, i, j;
unsigned    rev_initial, ss_contents, ms_contents;
float        chanhilim;
double      denomin, tanang, tansqr;
initial=1;
mark_1=0;       /* initialize excessdata marker */
num_wrap=0;     /* initialize excess data counter */
k=0;            /* initialize data counter */
l=0;            /* initialize buffer counter */
/*  menu label and initial value (note need 351 to get correct 537) */
     if ( !lwblabelmenu( 1, "OCTAL LAW" ) ) return( 0 );
     if ( !lwbupdatemenu( 1, "%o", law ) ) return( 0 );
     if ( !lwblabelmenu( 2, "Initial Load" ) ) return( 0 );
     if ( !lwbupdatemenu( 2, "%d", initial ) ) return( 0 );
     if ( !lwblabelmenu( 3, "Phase Angle" ) ) return( 0 );
     if ( !lwbupdatemenu( 3, "%f", iangle ) ) return( 0 );
```

```
    if ( !lwblabelmenu( 4, "Sample Freq" ) ) return( 0 );
    if ( !lwbupdatemenu( 4, "%f", dfs) ) return( 0 );
       /* compute demodulates per digit */
    dem_dig = (int)((dfs / dfc)*cycles_dig);
    if ( !lwblabelmenu( 5, "Threshold" ) ) return( 0 );
    if ( !lwbupdatemenu( 5, "%f", thresh_fact) ) return( 0 );

/* check input data */
    /* get dimensionality and size of input message data */
    lwbgetparam( inA, 0, LWBMSGDIM, &indimA );
    lwbgetparam( inB, 0, LWBMSGDIM, &indimB );
    lwbgetparam( inA, 0, LWBMSGSIZES, insizesA );
    lwbgetparam( inB, 0, LWBMSGSIZES, insizesB );
    if ( indimA == 2 )      counts = insizesA[0] * insizesA[1];
    else if ( indimA == 1 )counts = insizesA[0];
    /* check input data type */
    lwbgetparam( inA, 0, LWBMSGTYPE, &intypeA );
    if ( intypeA != 5 )
    {
        lwberror( ERROR, "Input A message data type must be FLOAT" );
        return( 0 );
    }
    lwbgetparam( inB, 0, LWBMSGTYPE, &intypeB );
    if ( intypeB != 5 )
    {
        lwberror( ERROR, "Input B message data type must be FLOAT" );
        return( 0 );
    }
    /* get gain, conv_gain, slope, and baseline data from msg header */
    lwbgetparam( inA, 0, LWBCHANGAIN, &gain );
    lwbgetparam( inA, 0, LWBCHANCONVGAIN, &conv_gain );
    lwbgetparam( inA, 0, LWBCHANSLOPE, &slope );
    lwbgetparam( inA, 0, LWBCHANBASELINE, &baseline );
/* compute sequence length (digits/seq.) and polynomial degree */
    temp=law;
    seq_len=1;
    rev_initial = 0;
    degree = 0;
       while (temp>>=1)
    {
        seq_len <<= 1;
        degree++;
    }
/* compute scram and unscram arrays */
    end_bit=seq_len-->>1;
    for (i=0; i < degree; i++)
    {
        rev_initial = (rev_initial << 1) | (initial&1);
        initial >>= 1;
    }
    ms_contents = 1;
```

```
    ss_contents = rev_initial;
    for (i = 0; i < seq_len; i++)
    {
        scram[i] = ss_contents;
        unscram[i] = ms_contents;
        temp = ss_contents & law;
        ss_contents >>= 1;
        do {if (temp &1) ss_contents^=end_bit;} while (temp >>=1);
            if (ms_contents &1)
                ms_contents = (ms_contents^law) >> 1;

            else
                ms_contents >>= 1;
    }
/* compute dc bias removal factors to be used in run */
    angle = iangle * PI / 180.0;
    tanang = tan(angle);
    tansqr = tanang * tanang;
    denomin = ((double)seq_len * (double)seq_len + tansqr);
    fact_r = (tansqr - (double)seq_len) / denomin;
    fact_i = ((double)seq_len + 1.0) * tanang / denomin;
/* prepare output message */
    /* copy input general info to output message */
    if ( !lwbnewmsg( &outmsg ) ) return( 0 );
    if ( !lwbcopymsg( inA, outmsg ) ) return( 0 );
    /* change output data size */
    osizes[0]= seq_len * dem_dig;
    osizes[1]= 1;
    osizes[2]= 0;
    lwbsetparam( outmsg, 0, LWBMSGSIZES, osizes );
    /* copy input message range info to output message */
    if ( !lwbnewrange( outmsg, 1 ) ) return( 0 );
    if ( !lwbcopyrange( inA, outmsg ) ) return( 0 );
/*  set new domain parameters */
    if ( !lwbnewdomain( outmsg ) ) return( 0 );
    if ( !lwbcopydomain( inA, outmsg ) ) return( 0 );
    if (!lwbgetparam( outmsg, -1, LWBCHANHILIM, &chanhilim)) return(0);
    chanhilim *= seq_len;
    if (!lwbsetparam( outmsg, -1, LWBCHANHILIM, &chanhilim)) return(0);
    lwbgetparam( outmsg, 0, LWBMSGSIZES, osizes );
    /* create a data buffer for output message */
    if ( !lwbnewbuf( outmsg ) ) return( 0 );
    if ( !lwbgetdatbuf( outmsg, &outdata ) ) return( 0 );
    /* send message on its way and release the input */
    lwbwrite( outmsg, out );
    lwbrelease( inA );
    lwbrelease( inB );
    return( 1 );
}
/************************************ RUN ****************************/
ufrun( inA, inB, out )
```

```
int inA, inB, out;
{
int        m,              /* index counter for stp */
          hadamard(),      /* hadamard transform routine */
          *scram_ptr,      /* pointer to scrambled data array */
          *unscram_ptr,    /* pointer to unscrambled data array */
          stp,             /* data items/digit since must do hadamard
                              correlation once for every data item
                              (ie 20*2 times for re & im parts @ 228
hz) */
          i,               /* general counter index */
          j,               /* general counter index */
          cor_r,           /* real correction factor */
         cor_i,           /* imag correction factor */
          *ptr,            /* ptr to main data array (dec_data) */
          *data_ptr,       /* ptr to current data item of digit
                              (ie 1st,or 2nd, etc) */
          *move_ptr,       /* ptr to temp array of seq_len digits to be
                              passed to hadamard routine (ie all the
1st,
                              or 2nd, etc data items of each digit */
          *sum_ptr;        /* ptr to seq_sum array which holds the 0th
                              item of each sequence passed to hadamard
                              to accumulate bias */
double     xt, yt, sqrt(), r_data, i_data;
float      out_sum, out_ave, pts_sum;
/* sanity check */
   if ( !inA || !outmsg ) return( 0 );
   if ( !inB || !outmsg ) return( 0 );
   if ( !lwbgetparam( outmsg, 0, LWBMSGLINKS, &links ) ) return( 0 );
   if ( links > 0 )
   {
      lwbunget( inA );
      lwbunget( inB );
      return( 1 );
   }
   if ( !lwbgetdatbuf( inA, &indataA ) ) return( 0 );
   if ( !lwbgetdatbuf( inB, &indataB ) ) return( 0 );
  /* enter demodulates into dec_data array one buffer at a time
      (r0,i0,r1,i1,.....r19,i19) until a full sequence is recieved
      (dem_dig*seq_len*2 = 10200 data items for law537 @ 228hz)
      (r0,i0,...r10199,i10199) */
   if (k < (seq_len * dem_dig * 2 /step)) /* ie if less than a full
sequence
                                             has been recieved */
   {
      if ((num_wrap >0) && (k==0)) /* if this is 1st buffer of a new seq
                                      and there was a remainder from the
                                      last sequence */
      {
         for ( i = 0; i < num_wrap; i += step)
```

```
      {
         dec_data[k++] = dec_data[mark_1++];
         dec_data[k++] = dec_data[mark_1++];
       }
    }
    for ( i = 0; i < counts; i += step)
    {
       dec_data[k++] = (long)(1e7*indataA[i]); /* note: mult by number
                                                  to get large enough to
                                                  convert to long int */
      dec_data[k++] = (long)(1e7*indataB[i]);
      if(k == (seq_len * dem_dig * 2 / step))
       {
       mark_1 = k;    /* set marker for first data of next sequence */
       num_wrap=((counts-i)-1); /* number of data to wrap to next seq*/
      }
    }
}
if (k >= (seq_len * dem_dig * 2 /step)) /* after a full sequence
                                           is received... */
{
  ptr = dec_data;              /* ptr to main data array (dec_data) */
  sum_ptr = seq_sum;           /* assign sum_ptr to seq_sum array */
  move_ptr = cbuf;             /* assign move_ptr to cbuf array */
  stp = dem_dig * 2;           /* define stp as data items per digit
                                  (real & im) */
     /* Take the nth data element of each digit in the m-sequence,
        scramble them, perform hadamard transform, and then unscramble
        them.  Repeat for each successive element in the digits. */
  for (m = 0; m < stp; m++)
  {
     *move_ptr = 0;       /* initial entry is zero */
     data_ptr = ptr + m;
     scram_ptr = scram;
     for (i=0; i < seq_len; i++)
     {
       *(move_ptr + *scram_ptr++) = *data_ptr;   /* get data values */
       data_ptr += stp;
     }
      hadamard(degree, move_ptr);
     *sum_ptr++ = *move_ptr;
     data_ptr = ptr + m;
     unscram_ptr = unscram;
     for (i=0; i < seq_len; i++)
     {
       *data_ptr = *(move_ptr + *unscram_ptr++); /* put results back*/
       data_ptr += stp;
     }
   }
  for (i=0; i < dem_dig; i++)    /*  bias removal loop  */
  {
```

```
            xt = (double)seq_sum[i+i];
            yt = (double)seq_sum[i+i+1];
            cor_r = (long)(xt * fact_r - yt * fact_i);
            cor_i = (long)(xt * fact_i + yt * fact_r);
            data_ptr = ptr + i + i;
            for (j = 0; j < seq_len; j++)
               {
               *data_ptr++ -= cor_r;
               *data_ptr-- -= cor_i;
               data_ptr += (dem_dig << 1);
               }
         }
         /* write to output buffer */
         j=0;
         for (i=0; i < seq_len * dem_dig; i++)
         {
            r_data =  (double)(dec_data[j++]*1e-7);
            i_data =  (double)(dec_data[j++]*1e-7);
            outdata[i] = (float)((r_data*r_data)+(i_data*i_data));
             /* if ( dx ) printf( "decoder outdata[6]=%f\n", outdata[6]);*/
            out_sum += outdata[i];
         }
         pts_sum = (float)(seq_len * dem_dig);
         out_ave = out_sum / pts_sum;
         if a point exceeds thresh_fact * out_ave, print buffer#, out_ave
               seconds, and value
         for (i=0; i < seq_len * dem_dig; i++)
         {
            if (outdata[i] > out_ave * thresh_fact)
            {
               printf("buffer #%d   ",l);
               printf("ave=%4.4e v.   ", out_ave);
               printf("%4.3f sec. = %4.4e v.\n", i / dfs, outdata[i]);
            }
         }
         /* write output data message */
         lwbwrite( outmsg, out );
         k = 0;
         l = l+1;
      }
      /* release input message */
      lwbrelease( inA );
      lwbrelease( inB );
      return( 1 );
   }
   /  *    H  A  D  A  M  A  R  D    F  H  T    r  o  u  t  i  n  e
   **************************************************/
   #include <math.h>
   hadamard(n, data)
   int     n, *data;
   {
```

```
int      *ptrp, *ptr1, *ptr2, temp1, temp2;
unsigned int two_step, parts, part;
register unsigned int step, k;
step = (parts = 1) << n;
while ((step = (two_step = step) >> 1) != 0)
{
     ptrp = data;
     part = parts;
     do
     {
          ptr1 = ptrp;
          ptrp += two_step;
          k = step;
          do
          {
              ptr2 = ptr1 + step;
              *ptr1 = (temp1 = *ptr1) + (temp2 = *ptr2);
              *ptr2 = temp1 - temp2;
              ptr1++;
          } while (--k);
     } while (--part);
     parts <<= 1;
}
return;
}


/************************************** STOP **********************/
ufstop( inA, inB, out )
int inA, inB, out;
{
     if ( !inA || !out ) return( 0 );
     if ( !inB || !out ) return( 0 );
     lwbwrite( outmsg, out );
     lwbrelease( inA );
     lwbrelease( inB );
     return( 1 );
}
/************************************ MENU ************************/
ufmenu( key )
int key;
{
      if ( key == 1 )
      {
            lwbgetint( "Enter DECIMAL Law (11,351,827,1051,3047)",11,3047,
                 &law );
          if ( !lwbupdatemenu( 1, "%o", law ) ) return( 0 );
     }
     if ( key == 2 )
     {
```

```
            lwbgetint( "Enter Initial Register Load (not zero)", 1, 10000,
                        &initial );
            if ( !lwbupdatemenu( 2, "%d", initial ) ) return( 0 );
        }
        if ( key == 3 )
        {
            lwbgetdouble( "Enter Phase Angle", 0.0, 180.0, &iangle);
            if ( !lwbupdatemenu( 3, "%f", iangle ) ) return( 0 );
        }
        if ( key == 4 )
        {
            lwbgetfloat("Enter Sample Freq", 1.0, 1000.0, &dfs);
            if ( !lwbupdatemenu( 4, "%f", dfs) ) return( 0 );
        }
        if ( key == 5 )
        {
            lwbgetfloat("Enter    threshold    factor",    0.0,    100000.0,
&thresh_fact);
            if ( !lwbupdatemenu( 5, "%f", thresh_fact ) ) return( 0 );
        }
        return( 1 );
}
/************************************  DROP  ************************/
ufdrop()
{
        if ( !lwbfreemsg( &outmsg ) ) return( 0 );
        return( 1 );
}
```

```
/*************************************************************************
SPEWRIT.C
This program takes output from the power spectrum module and finds the
highest peak in a 5 knot doppler window. It writes the peak frequency and
amplitude (v²/Hz) to a file.  It also calculates the mean noise in a 24 Hz
band centered at 57 Hz and writes it to a file.
*************************************************************************/
static char SccsID[] = "@(#)uftest.c8.1 (MASSCOMP) 2/5/90";
#include "lwbufunc.h"
#include "errors.h"
#include <stdio.h>
FILE    *outfile1, *outfile2, *outfile3;
char    outfilenam1[15]="m",  outfilenam2[15]="f", outfilenam3[15]="n";
float   *idata, thresh;
int  itype=0, idimension=0, insizes[3]={0,0,0}, count=0, dx=0;
static int m;
/*************************************************************************
 * >> Entry point from LWB — must be first function defined in file << *
 *************************************************************************/
uftest( action, inA, inB, inC, inD, outX, outY, debug )
int *action, *inA, *inB, *inC, *inD, outX, outY, *debug;
{
     int status;
     dx = *debug;
     switch ( *action )
     {
          case SETUPCMD:status = ufsetup( *inA ); break;
          case RUNCMD:  status = ufrun( *inA ); break;
          case STOPCMD:status = ufstop( *inA ); break;
          case MENUCMD:status = ufmenu( *inA ); break;
          case DROPCMD:status = ufdrop(); break;
          default:      break;
     }
     return( status );
}
/******************************** SETUP **********************************/
ufsetup( in )
int in;
{
        m=1;
/* set up menu label and initial value  */
     if ( !lwblabelmenu( 1, "MAG filename" ) ) return( 0 );
     if ( !lwbupdatemenu( 1, "%s", outfilenam1) ) return( 0 );
     if ( !lwblabelmenu( 2, "FREQ filename" ) ) return( 0 );
     if ( !lwbupdatemenu( 2, "%s", outfilenam2) ) return( 0 );
     if ( !lwblabelmenu( 3, "NOISE filename" ) ) return( 0 );
     if ( !lwbupdatemenu( 3, "%s", outfilenam3) ) return( 0 );
      if ( !lwblabelmenu( 4, "Threshold" ) ) return( 0 );
      if ( !lwbupdatemenu( 4, "%f", thresh) ) return( 0 );
/* check input messages */
     /* get dimensionality of input message data */
```

```
     lwbgetparam( in, 0, LWBMSGDIM, &idimension );
     if ( dx ) printf( "spewrit idimension=%d\n", idimension );
     /* get size of input message data */
     lwbgetparam( in, 0, LWBMSGSIZES, insizes );
     if (dx) printf("demod insizes=%d,%d,%d\n", insizes[0], insizes[1],
                          insizes[2] );
     count = insizes[0];
     if ( dx ) printf( "spewrit count=%d\n", count );

     /* check input data type */
     lwbgetparam( in, 0, LWBMSGTYPE, &itype );
     if ( dx ) printf( "demod itype=%d\n", itype );
     if ( itype != 5 )
     {
          lwberror( ERROR, "Writer input data type must be FLOAT" );
          return( 0 );
     }
/* open output files */
        if((outfile1 = fopen(outfilenam1,"a+")) == NULL)
        {
          lwberror( ERROR, "Couldn't open the MAG file" );
          return( 0 );
        }
        if((outfile2 = fopen(outfilenam2,"a+")) == NULL)
        {
          lwberror( ERROR, "Couldn't open the FREQ file" );
          return( 0 );
       }
        if((outfile3 = fopen(outfilenam3,"a+")) == NULL)
        {
          lwberror( ERROR, "Couldn't open the NOISE file" );
          return( 0 );
       }
/* release the input */
     lwbrelease( in );
     return( 1 );
}
/*************************** RUN ************************************/
ufrun( in )
int in;
{
 int  i, bin;
float max =0.0, freq, noise=0.0, nave=0.0, snr=0.0;
     if ( !lwbgetdatbuf( in, &idata ) ) return( 0 );
/* process data */
      for (i=8179; i < 8208; i++)
      {
         if(idata[i] > max)  /* get max amplitude and it's freq in +/-
                                5kts band about 57.0 Hz */
         {
            max = idata[i];
```

```
            freq = (float)i * 0.0069585;
            bin = i;
          }
        }
        for (i=6469; i < 9917; i++)
        noise = noise + idata[i]; /* get noise power in 24 Hz passband */
        nave=noise / 3447;
        max =   (max   / 1167.40902475);  /* convert from LWB to reality */
        noise = (noise / 1167.40902475);
        noise = noise - max;
        snr = max / noise;
        printf("bin=%d  %fhz  m=%e n=%4.4f \n",bin,freq,max,noise);
        printf("    snr=%4.4f   ",snr);
        fprintf(outfile1,"%4.4e ",max);
        fprintf(outfile2,"%4.4f ",freq);
        fprintf(outfile3,"%4.4e ",noise);
        printf("finished row number %d\n",m++);
      lwbrelease( in );
      return( 1 );
}
/****************************  STOP  ********************************/
ufstop( in)
int in;
{
      if ( !in ) return( 0 );
      lwbrelease( in );
       printf("\n");
       fprintf(outfile1,"\n");
       fprintf(outfile2,"\n");
       fprintf(outfile3,"\n");
      fclose(outfile1);
      fclose(outfile2);
      fclose(outfile3);
       printf("number of rows =%d\n",m-1);
      return( 1 );
}
/****************************  MENU  ********************************/
ufmenu( key )
int key;
{
      if ( key == 1 )
      {
           lwbgetstring( "Enter MAG Filename", 1, 15, outfilenam1 );
           if ( !lwbupdatemenu( 1, "%s", outfilenam1) ) return( 0 );
      }
      if ( key == 2 )
      {
           lwbgetstring( "Enter FREQ Filename", 1, 15, outfilenam2 );
           if ( !lwbupdatemenu( 2, "%s", outfilenam2) ) return( 0 );
      }
      if ( key == 3 )
```

```
    {
        lwbgetstring( "Enter NOISE Filename", 1, 15, outfilenam3 );
        if ( !lwbupdatemenu( 3, "%s", outfilenam3) ) return( 0 );
    }
        if ( key == 4 )    /* not currently used */
        {
            lwbgetfloat( "Enter Output Threshold", 1.0, 1000.0,
&thresh);
            if ( !lwbupdatemenu( 3, "%f", thresh ) ) return( 0 );
        }
    return( 1 );
}
/******************************** DROP *********************************/
ufdrop()
{
    return( 1 );
}
```

REFERENCES

1. Bendat Julius S. and Piersol Allan G., *Random Data: Analysis and Measurement Procedures*, 2nd ed., pp. 385 - 391, John Wiley & Sons, 1986.

2. Birdsall, T., *Estimates of Signal to Noise Ratio and Integration Times for Heard Island Feasibility Experiment*, CSPL, University of Michigan, 20 November, 1990.

3. Birdsall, T.G., Metzger, K., and Spindel, Robert C., "Signal Processing for Ocean Tomography with Moving Ships", *Asilomar Conf. on Signals, Systems, and Comp.*, Monterey, CA, November 1, 1988.

4. Birdsall, T.G., and Metzger, K., "Signal Processing for Moved Source/Receiver Tomography (MSR) aka Moving Source Tomography (MOST)", unpublished manuscript, August, 1987.

5. Birdsall, T.G., and Metzger, K., "Signal Schedule for the Heard Island Feasibility Experiment," CSPL, University of Michigan, 13 August 1990.

6. Borish, Jeffrey and Angell, James, "An Efficient Algorithm for Measuring the Impulse Response Using Pseudorandom Noise," *J. Audio Engineering Society*, v.31, No.7, pp. 478 - 488, July/August 1983.

7. Burdic, William S., *Underwater Acoustic System Analysis*, pp. 17 - 154, Prentice-Hall, 1984.

8. Clay, Clarence S. and Medwin, Herman, *Acoustical Oceanography*, pp. 29 - 136, John Wiley & Sons, 1977.

9. Cohn, Martin and Lempel, Abraham, "On Fast M-Sequence Transforms," *IEEE Transactions on Information Theory*, pp. 135 - 137, January 1976.

10. Dees, Robert C., *Signal Processing and Preliminary Results in the 1988 Monterey Bay Tomography Experiment*, MS Thesis, Naval Postgraduate School, Monterey, CA, June 1989.

11. Eldred, Randy M., *Doppler Processing of Phase Encoded Underwater Acoustic Signals*, MS Thesis, Naval Postgraduate School, Monterey, CA, September 1990.

12. Flatte, Stanley, and others, *Sound Transmission through a Fluctuating Ocean*, pp. 3 - 61, Cambridge University Press, 1979.

13. Hansen, J., and Lebedeff, S., "Global Trends of Measured Surface Air Temperature," *Journal of Geophysical Research*, v. 92, pp. 13345 - 13372, 1987.

14. Jones, R.M., Riley, J.P., and Georges, T.M., "HARPO a Versatile Three-Dimensional Hamiltonian Ray-tracing Program for Acoustic Waves in an Ocean with Irregular Bottom," *Wave Propagation Lab. NOAA*, Boulder, Colorado, 457 pp., 1986.

15. Keeling, C., and others, "A Three-Dimensional Model of Atmospheric CO_2 Transport Based on Observed Winds," *Geophysical Monograph*, v.55, pp. 165 - 236, 1989.

16. Kinsler, L., and others, *Fundamentals of Acoustics*, 3rd ed., pp. 117 - 120, John Wiley & Sons, 1982.

17. Manabe S. and Stouffer, R.J., "Sensitivity of a Global Climate Model to an Increase of CO_2 Concentration in the Atmosphere," *Journal of Geophysical Research*, v.85, pp. 5529 - 5554, 1986.

18. Munk, W., and Forbes, A., "Global Ocean Warming: An Acoustic Measure?," *Journal of Physical Oceanography*, v. 19, pp. 1765 - 1778, 1989.

19. Munk, W., *The Heard Island Experiment*, National Academy Press, 1990.

20. National Research Council, *Changing Climate: Report of the Carbon Dioxide Assessment Committee*, National Academy Press, 1983.

21. Neftel, A., Moor, E., Oeschger, H., and Stauffer, B., "Evidence From Polar Ice Cores for the Increase in Atmospheric CO_2 in the Past Two Centuries," *Nature*, v.315, pp. 45 - 47, 1985.

22. Ort, C., *Spatial and Temporal Variability of Cross Basin Acoustic Raypaths*, MS Thesis, Naval Postgraduate School, Monterey, CA, December 1990.

23. Semptner, A.J., and Chervin, R.M., "A Simulation of the Global Ocean Circulation with Resolved Eddies," *Journal of Geophysical Research*, v.93, pp. 15502 - 15522, 1988.

24. Semtner, J.J., and Chervin, R.M., "Environmental Effects on Acoustic Measures of Global Ocean Warming," *Journal of Geophysical Research*, v. 95, pp. 12973-12982, 1990.

25. Spindel, Robert C., "Signal Processing in Ocean Tomography," *Adaptive Methods in Underwater Acoustics*, ed. H.G. Urban, pp. 687 - 710, D.Reidel Publishing Company, 1985.

26. Spindel, Robert C., "Ocean Acoustic Tomography: A Review," *Current Practices and New Technology in Ocean Engineering*, v.11, pp. 7 - 13, 1986.

27. Thorp, W. H., "Analytic Description of the Low Frequency Attenuation Coefficient," *Journal of the Acoustical Society of America*, v.42, pp. 270, 1967.

28. Van Trees, H., *Detection, Estimation and Modulation Theory, Part I*, pp. 273 - 287, John Wiley & Sons, 1968.

29. Welch, P.D., "The use of the Fast Fourier Transform for the Estimation of Power Spectra," *IEEE Transactions on Audio and Electroacoustics*, v. AU-15, pp. 70-73, June 1970.

30. Ziemer, Rodger E. and Peterson, Rodger L., *Digital Communications and Spread Spectrum Systems*, pp. 365 - 415, Macmillan Publishing Company, 1985.

INITIAL DISTRIBUTION LIST

	Copies	No.
1.	Defense Technical Information Center Cameron Station Alexandria, VA 22304-6145	2
2.	Library, Code 52 Naval Postgraduate School Monterey, CA 93943-5002	2
3.	Chairman (Code OC/Co) Department of Oceanography Naval Postgraduate School Monterey, CA 93943-5000	1
4.	Director Naval Oceanography Division Naval Observatory 34th and Massachusetts Avenue NW Washington, DC 20390	1
5.	Commander Naval Oceanography Command Stennis Space Center MS 39529-5000	1
6.	Commanding Officer Naval Oceanographic Office Stennis Space Center MS 39522-5001	1
7.	Commanding Officer Naval Oceanographic and Atmospheric Research Laboratory Stennis Space Center MS 39529-5004	1
8.	Chief of Naval Research 800 N. Quincy Street Arlington, VA 22217	1
9.	Library Scripps Institution of Oceanography P. O. Box 2367 La Jolla, CA 92037	1

10. Library 1
Department of Oceanography
University of Washington
Seattle, WA 98105

11. Library 1
College of Oceanography
Oregon State University
Corvallis, OR 97331

12. Commander 1
Oceanographic Systems Pacific
Box 1390
Pearl Harbor, HI 96860

13. Prof. James H. Miller, Code EC/Mr 2
Department of Electrical and Computer Engineering
Naval Postgraduate School
Monterey, CA 93943

14. Prof. Ching-Sang Chiu, Code OC/Ci 1
Department of Oceanography
Naval Postgraduate School
Monterey, CA 93943

15. LCDR Gary R. Frogner 2
262 Frogner Rd
Chehalis, WA 98532

16. Dr. Keith Von der Heydt 1
Department of Applied Ocean Physics and Engineering
Woods Hole Oceanographic Institution
Woods Hole, MA 02543

17. Dr. Arthur B. Baggeroer 1
Department of Ocean Engineering
Massachussetts Institute of Technology
Cambridge, MA 02139

18. Dr. K. Lashkari 1
Monterey Bay Aquarium Institute
160 Central Avenue
Pacific Grove, CA 93950

19. Dr. M. Briscoe 1
Office of Naval Research
800 N Quincy Street
Arlington, VA 22217-5000

20. Dr. Walter Munk 1
Institute of Geophysics and Planetary Physics, A-025
Scripps Institute of Oceanography
University of California, San Diego
La Jolla, CA 92093

21. Dr. Andrew Forbes 1
CSIRO Division of Oceanography
GPO Box 1538
Hobart, Tasmania 7001
Australia

22. Dr. Kurt Metzger, Jr. 1
Communications and Signal Processing Laboratory
Department of Electrical Engineering and Computer Science
University of Michigan
Ann Arbor, MI 48109

23. Dr. Theodore G. Birdsall 1
Communications and Signal Processing Laboratory
Department of Electrical Engineering and Computer Science
University of Michigan
Ann Arbor, MI 48109

24. Dr. Peter Mikhalevsky 1
SAIC
4507 N 26th St.
Arlington, VA 22207

25. Dr. R.C. Spindel 1
Director, Applied Physics Laboratory
University of Washington
1013 NE 40th St.
Seattle, WA 98105

26. Dr. Mercer 1
Applied Physics Laboratory
University of Washington
1013 NE 40th St.
Seattle, WA 98105

Thesis
F8964 Frogner
c.1 Monitoring of global
acoustic transmissions.

Thesis
F8964 Frogner
c.1 Monitoring of global
acoustic transmissions.

DUDLEY KNOX LIBRARY
3 2768 00034125 9